Univerza na Primorskem

Fakulteta za matematiko, naravoslovje in informacijske tehnologije

Andrej Cimperšek

Razvoj sistema za centraliziran nadzor strežnikov

Zaključna naloga

Koper, junij 2011

Mentor: doc. dr. Peter Rogelj

Kazalo

Slike

Tabele

Povzetek

V tej nalogi je predstavljeno načrtovanje informacijskega sistema za centraliziran nadzor strežnikov. Predstavljene je večina faz razvoja, z izjemo vzdrževanja. Začetna poglavja opisujejo načrtovanje sistema, zadnji dve pa implementacijo in testiranje. Implementacija je izvedena v programskem jeziku Python in za hranjenje podatkov uporablja podatkovno bazo MongoDB. Vsa uporabljena programska oprema, z izjemo načrtovalskega orodja Microsoft Visio, je brezplačna in prostodostopna. Sistem bo omogočal nadzor nad zasedenostjo diskov in pomnilnika, nadzor nad procesi in izvajanje sistemski skript.

Abstract

In this paper we present development of an information system. All stages with exception of maintenance are presented. The initial chapters describe the planning system, the last two implementation and testing. Components are written in Python programming language and data is stored into MongoDB database. All used software, with exception of Microsoft Visio, is free and open source. This system will offer central monitoring of disks and memory usage, monitoring of processes and execution of system scripts.

Zahvala

Rad bi se zahvalil mentorju za ves trud, ki ga je vložil v pregledovanje pojmov in idej prisotnih v tej nalogi.

Slovarček

Slovarček vsebuje nekatere pojme, ki se pojavljajo v nalogi.

Python	Programski jezik.
SSL (*Secure Sockets Layer*)	Protokol, ki omogoča šifrirano povezavo med strežnikom in odjemalcem.
nit (angl. *thread*)	Najmanjša enota opravil. Več niti lahko izvajamo istočasno, če strojna oprema to podpira.
programski jezik (angl. *programming language*)	Umetni jezik, ki se uporablja za pisanje računalniških programov.
operacijski sistem (angl. *operating system*)	Skupek sistemskih programov, ki omogoča delovanje računalniškega sistema.
računalniški program (angl. *computer program*)	Zaporedje ukazov, ki jih lahko izvede računalnik.
centralni nadzorni strežnik (angl. *central monitoring server*)	Strežnik, ki nadozruje nadzorovane strežnike.
nadzorovan strežnik (angl. *monitored server*)	Strežnik, ki ga nadzorujemo s centralnim nadzornim strežnikom.
API (*Application Programming Interface*)	Skupek pravil in specifikacij, ki programski opremi omogočajo medsebojno komunikacijo.
URL (*Uniform Resource Locators*)	Naslov spletnih strani v spletu.

Poglavje 1

Uvod

V zadnjih letih se je močno razvila in hkrati povečala uporaba virtualizacije v računalništvu. V preteklosti, ko virtualizacija ni bila tako razvita, so strežniki poganjali le en operacijski sistem, na katerem je teklo več storitev. Virtualizacija je v računalništvo prinesla povsem drugačno razmišljanje o uporabi in upravljanju strežnikov. Tako lahko en fizični strežnik poganja poljubno število virtualnih strežnikov, ki se obnašajo enako kot da bi bili fizični. Posledica tega je, da so zaradi varnosti, izoliranost in ostalih razlogov, začele posamezne storitve izvajati na ločenih virtualnih strežnikih. Če so prej na enem strežniku tekli tako spletni strežnik, poštni strežnik, baze in ostalo, se zdaj raje odločamo za ločevanje posameznik sistemov. S tem pa nastane nov problem - upravljanje z velikim številom fizični in virtualnih strežnikov. V tej nalogi je opisano načrtovanje takega informacijskega sistema in primer implementacije.

Cilj te naloge je predstaviti načrtovanje informacijskega sistema. Naloga skuša predstaviti vse vidike programskega inženirstva na praktičnem primeru, z izjemo vzdrževanja.

Naloga je sestavljena iz sedmih delov. Vsak del predstavlja eno od faz prgoramskega procesa. V prvem je predstavljena definicija problema. V drugem delu je opravljena študija izvedljivosti. Tretji del predstavlja analiza in definicija zahtev. V četrtem delu je načrtovanje sistema. Peti del prestavlja načrtovanje programa. V šestem delu je predstavljena izvedba programa. V zadnjem delu, pa je predstavljeno testiranje programa.

Poglavje 2

Definicija problema

Problem nadzora in upravljanja večjega števila strežnikov predstavlja velik iziv upraviteljem takih sistemov. Podatkovna središča z nekaj sto ali tisoč strežniki so postali nekaj čisto običajnega. Ročno upravljanja vsakega posamezno tako ne pride v poštev, saj bi zahtevalo preveč človeškega dela, kar hkrati tudi pomeni večjo možnost napake. V podatkovnem središču s sto strežniki bi moralo biti zaposlenih več sistemskih administratorjev, ki bi opravljali dolgočasna ponavljajoča se opravila. To je tako ekonomsko kot časovno neopravičljivo. Če bi obstajal sistem za lažje upravljanje in nadzor večjega števila strežnikov hkrati bi lahko ta sredstva porabili v produktivnejše in z ekodomskega vidika boljše namene.

Problem, ki ga opisujemo predstavlja nadzor množice strežnikov, ki se nahajajo v istem omrežju, na katerih se opravlja ponavljajoča se opravila. Največji zahtevi sta možnost ponovnega zagona oziroma zaustavitev poljubnega strežnika ter možnost izvedbe poljubne sistemske skripte. Prav tako je zaželjen pregled procesov, ki so zagnani na izbranem strežniku in možnost zaustavitve le teh. Isti izziv predstavlja tudi nadzor strojne opreme, konkretno zasedenosti posameznik diskov, ter zasedenost pomnilnika.

Podjetja in posamezniki so se tega problema lotili reševati na različne načine. Najpogostrejši in hkrati enostavnejši način je preko raznih sistemskih skript, ki se poženejo na večjem številu strežnikov hkrati. Slabost tega je, da je težko nadzirati napredek in uspešnost takih skript. Prav tako se te skripte razlikujejo med samimi sistemi in tudi njihov prenos je težaven v heterogenem okolju. Se pravi, da so take skripte v redu le v manjših, homogenih okoljih. V večjih in heterogenih okoljih je ta princip neupraben.

Zaradi slabosti predhodno opisanega sistema so se iskale boljše rešitve.

Poglavje 3

Študija izvedljivosti

Pri izvedljivosti imamo dve večji oviri. Prva ovira je budget, ki je v tem primeru ničen, saj gre za akademski projekt. Druga ovira pa je rok izdelave. Tako je poleg pravočasnega zaključka projekta glavi cilj, da se razen lastnega dela ne ustvari dodatnih stroškov. Iz tega razloga smo se odločili, da ba vsa uporabljena programska oprema brezplačna in če bo le mogoče tudi odprtokodna, tako kot bo odprtokoden naš sistem.

Ker gre za večji sistem ga je potrebno razdeliti na ločene komponente, ki skupaj predstavljajo celoto. Posamezne komponente je možno neodvisno testirati in implementirati. Tako lahko vzporedno razvijamo neodvisne komponente in skrajšamo čas razvoja. V našem primeru ta vidik ni toliko pomemben, saj ekipo sestavlja le en član. Za nas pomembenjši razlog je lažje obvladovanje sistema in preverba ustreznosti ter testiranje posamezne komponente.

Rok za zaključek projekta je sredina septembra 2011, zato smo časovno omejeni na približno dva meseca. V tem času je potrebno zaključiti tri večje komponente sistema. Te ključne komponente so centralni strežnik, spletni vmesnik in klient. Vse tri komponente so približno enako zahtevne. Za vsako izmed komponent imamo približno dva tedna časa za izdelavo, saj moramo upoštevati tudi čas potreben za testiranje in odpravljanje napak, ter pisanje zaključne naloge. Časovni roki so tako okvirno postavljeni in sistem mora biti implementiran do druge polovice avgusta 2011, kateremu sledi testiranje in odprava napak.

Zakonov v našem primeru ne bomo kršili, saj gre za sistem za interno uporabo in ne zlonamerno programsko opremo, z namenom pridobitve protipravnega premoženja.

Za izdelavo sistema želimo uporabiti dobro poznane standarde kjer je le mogoče. Tako lahko za hranjenje podatkov uporabimo podatkovno bazo, za izmenjavo podatkov dobro poznano protokol HTTP, na katerem temelji današnji splet. Za izdelavo spletnega vmesnika lahko uporabimo standardne spletne rešitve, konkretno HTML5, CSS in JavaScript. Zaradi dobre prenosljivosti in podpore vsem popularnim operacijskim sistemom se nagibamo k programskemu jeziku Python, kot izbiri za implementacijo celotnega strežniškega dela in klienta. Celoto pa bo poganjal brezplačen in odprtokoden operacijski sistem Linux. Vse omenjene tehnologije in protokoli so prosto dostopne, brezplačne in odprtokodne.

Poglavje 4

Analiza in definiranje zahtev

V tem poglavju bomo predstavili analizo in definicijo zahtev za sistem za centraliziran nadzor in upravljanje. Ta del navadno opravi sistemski analitik v sodelovanju z naročnikom oziroma uporabnikom. Ker pa gre v našem primeru za akademski projekt naročnik ne obstaja, oziroma smo naročnik mi sami. Končni uporabnik sistema bo sistemski administrator. Ker imamo možnost direktnega kontakta s končnim uporabnikom lažje izvemo realne potrebe in želje, hkrati pa težje izluščimo resnične potrebe.

4.1 Namen sistema

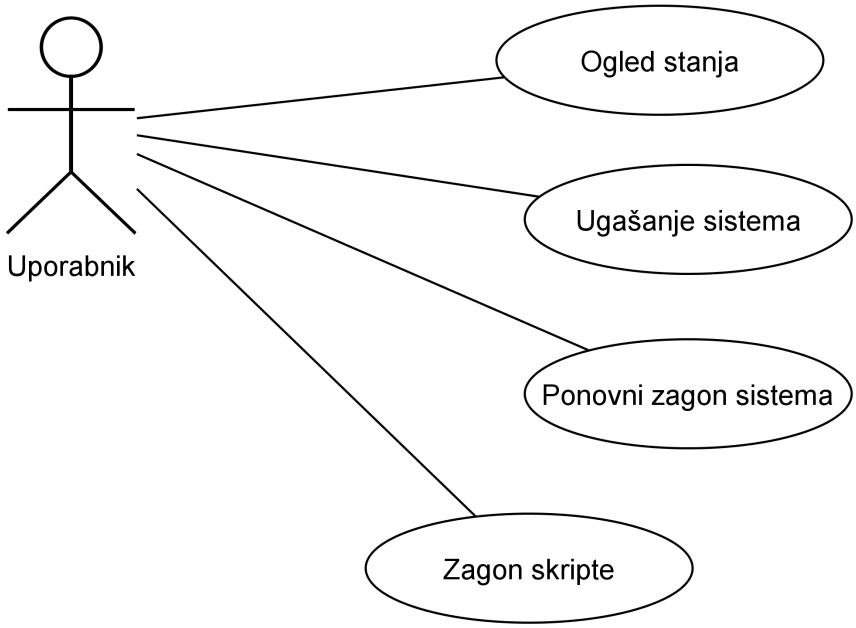
Sistem se razvija za uporabo v zaprtem krog ljudi znotraj podjetja. Zahteve izhajajo iz potreb po lažjem nadzoru večje količine strežnikov. Zahteve so sledeče:

- ogled stanja nadzorovanega strežnika
- ugašanje nadzorovanega strežnika
- ponovni zagon nadzorovanega strežnika
- zagon sistemske skripte

Z razvitjem sistema za rešitev teh zahtev bo uporabniku olajšan in pohitren nadzor in uporabljanje z večjim številom strežnikov.

4.2 Primeri uporabe

Spodnji diagram prikazuje primere uporabe sistema.



Slika 4.1: Primeri uporabe

Pred katerkoli akcijo na izbranem nadzorovanem strežniku se moramo v sistem prijaviti. Za to potrebujemo pravilno kombinacijo uporabniškega imena in gesla. Ko se v sistem uspešno prijavimo moramo v stranski drevesni strukturi izbrati željen nadzorovan strežnik. Po kliku na željen strežnik se nam odpre nov zavihek v katerem izvemo vse podatke in izvajamo akcije.

Ogled stanja predstavlja potrebno po ogledu porabe sredstev na sistemu. To zajema stanje zasedenosti trdih diskov in porabe pomnilnika. Do teh podatkov pridemo tako, da kliknemo na podzavihek Hardware na izbranem nadzornem strežniku. Tu se nam prikažejo informacije o tipu procesorja, o količini in zasedenosti pomnilnika ter informacije za vsak posamezen disk, ki vsebuje informacijo o imenu diska, velikosti in razpoložljivosti. Prav tako pod ogled stanja spadajo procesi, ki so pognani na nadzorovanem strežniku. O vsakem procesu lahko izvemo unikatni identifikator, ime, stanje, čas zagona in porabo sredstev.

Druga zahteva je ugašanje in ponovni zagon strežnika. To uporabniku omogoča enostaven centraliziran način za ugašanje sistemov, ki niso v uporabi. To storimo tako, da na podzavihku Info izbranega nadzorovanega strežnika kliknemo na gumb Restart za ponovni zagon, oziroma Shutdown za ugašanje.

Zagon skripte in ogled rezultatov izvedbe omogoča uporabniku, da na enostaven način pospeši dostavo in izvedbo poljubne aktivnosti na določen strežnik ali več teh. Skripta vrne izpis standardnega izhoda in standardne napake iz katerega je možno izvedeti uspešnost izvedbe in njen rezultat. Skripto lahko izdelamo tako, da kliknemo na podzavihek Script na izbranem nadzorovanem strežniku in izpolnemo vse potrebne podatke v obrazcu, ter zaključimo s klikom na Execute script.

Ko zaključimo z uporabo sistema se lahko iz sistema odjavimo s klikom na povezavo Logut v zgornjem desnem kotu spletnega vmesnika.

4.3 Varnost

Sistem mora biti varen pred nepooblaščenim dostop, ter pred pošiljanjem neveljavnih podatkov s strani nadzorovanega strežnika. Vstop v sistem mora biti zaščiten z uporabniškim imenom in geslom. To velja tako za nadzorovane kliente kot za uporabnike spletnega vmesnika. Nadzorovani klient se mora v sistem prijaviti ob vsakem pošiljanju in branju podatkov s svojo unikatno identifikacijsko številko in ključem. Uporabnik se mora v spletni vmesnik prijaviti samo enkrat, to je pred začetkom uporabe. Prijava uporabnika je veljavna dokler je veljavna seja, navadno dokler se ne zapre brskalnik. Omogočena mora biti tudi ročna odjava iz sistema.

4.4 Skalabilnost

Predvidena mora biti skalabilnost. To pomeni, da se v primeru povečanega obsega uporabe lahko centralni strežnik razširi z dodatnimi strežniki brez, da uporabnik ali nadzorovani strežnik to občuti.

Za prihodnost mora biti predvidena možnost implementacije sistema za obveščanje uporabnika. Ko imamo vse potrebne podatke na enem mestu lahko po potrebi izdelamo tudi sistem za obveščanje preko e-pošte, SMS sporočil ali drugačen način. Tak sistem lahko upravitelju samodejno sporoči, da se sistem obnaša nepredvidljivo ali da mu primanjkuje sistemskih sredstev.

Poglavje 5

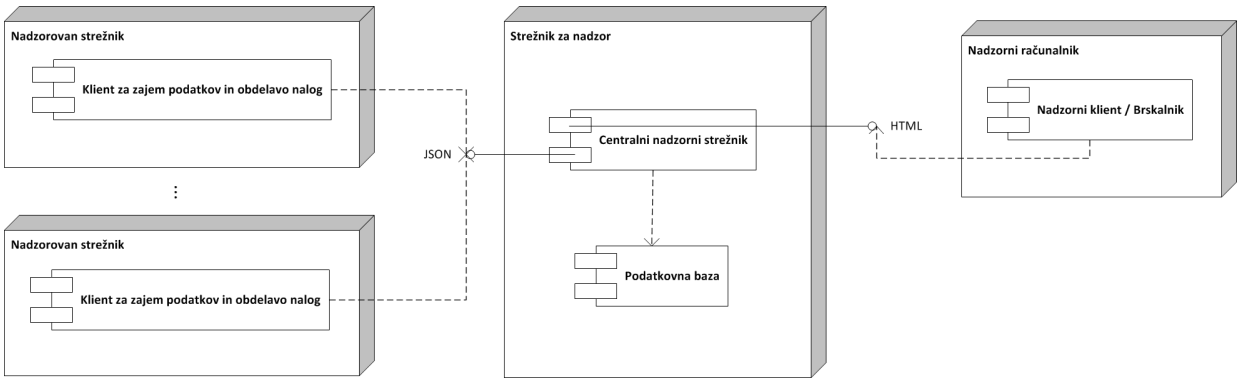
Načrtovanje sistema

Sistem, ki ga načrtujemo je zasnovan okoli centralnega strežnika. Ta strežnik prejema, hrani in obdeluje vse podatke. Ostali dve komponenti sta spletni vmesnik in klient za nadzor strežnika. Od strojne opreme zadostuje en računalnik priklopljen v omrežje. Strojne zahteve za strežnik so odvisne od količine strežnikov katere želimo upravljati oziroma nadzirati.

Iz definicije zahtev so razvidne ključne komponente sistema. To so centralni strežniški del, uporabniški vmesnik ter klient za nadzor. Ker do centralnega strežnika dostopa večja količina klientov za nadzor strežnika je ključno, da je ta del stabilen, hiter in odziven

5.1 Komponente in vmesniki

Komponente in vmesniki so razvidni iz spodnjega diagrama.



Slika 5.1: Komponente sistema in vmesniki

Centralni strežnik komunicira z bazo, ki navzven ni vidna uporabnikom in klientom za nadzor. Na voljo sta dva vmesnika. JSON je namenjen komunikaciji med klienti in centralnim strežnikom. Izbrali smo ga zaradi manjše kompleksnosti in manjše velikosti podatkov, ki jih prenašamo v primerjavi z XML zapisom. HTML vmesnik je namenjen spletnemu vmesniku oziroma uporabnikom.

Spletni HTML vmesnik

HTML vmesnika ne bomo specificirali saj gre za dobro poznan format, ki ga podpirajo vsi popularni spletni brskalniki. HTML se generira dinamično glede na podatke zapisane v podatkovni bazi.

JSON vmesnik

Klienti in centralni strežnik izmenujejo podatke preko JSON-a.

Oblika sporočila o strojni opremi:

```
{
  "ram": {
    "total": int,
    "free": int,
    "used": int
  },
  "cpu": [
    {
      "model": string,
      "bits": int,
      "number_ of_cores": int
    }
  ],
  "drive": [
    {
      "device": string,
      "mount": string,
      "free": int,
      "size": int
    },
    ...
  ]
}
```

Oblika sporočila o programski opremi:

```
{
    "python": {
        "implementation": string,
        "version": [
            string,
            string,
            string
        ]
    },
    "os": {
```

```
        "platform": string,
        "version": [
            string,
            string,
            string
        ],
        "system": string,
        "architecture": string
    }
}
```

Oblika sporočila o nalogah:

```
{
    "tasks": [{
        "id": string,
        "type": string,
        "created": datetime,
        "data": int/string/list
    }]
}
```

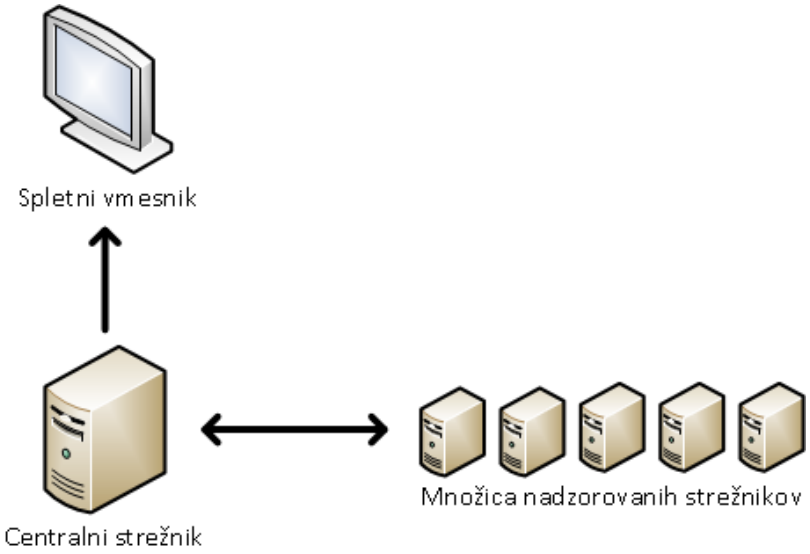
Oblika sporočila o procesih:

```
{
  "processes": [
    {
      "status": string,
      "name": string,
      "pid": int,
      "create_time": datetime,
      "user": string,
      "memory": [
        int,
        int
      ],
      "cpu": float
    }
  ]
}
```

5.2 Skalabilnost in zanesljivost

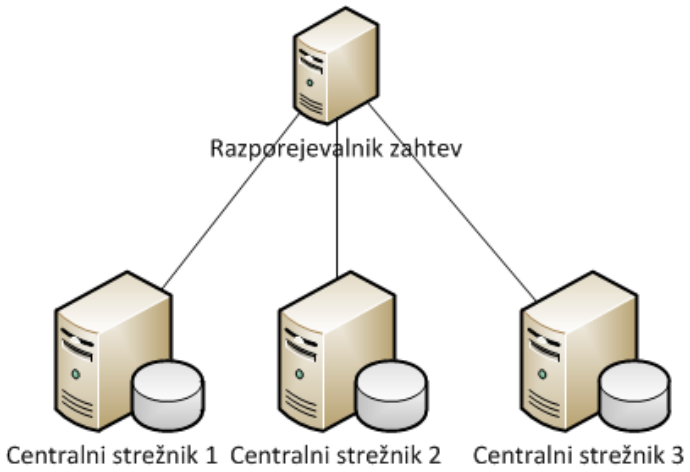
Zaradi varnosti podatkov in boljših zmogljivosti je priporočljivo, da se uporabi vsaj dva fizična ali virtualna strežnika za nudenje storitev, po možnosti na različnih fizičnih lokacijah. Priporočljivo je tudi, da so strežniki med seboj povezani s hitro mrežno povezavo. V

primeru uprabe dveh strežnikov, se na obeh namesti podatkovna baza, med njima pa ustvari povezava, ki omogoča replikacijo in uravnovešanje obremenitve. Podobno lahko storimo s spletnim strežnikom, le da med njiju postavimo še dodatno programsko opremo, ki skrbi za uravnovešanje zahtev na oba strežnika.



Slika 5.2: Arhitektura sistema

Če upoštevamo dejstvo, da je takih strežnikov lahko poljubno veliko, pridemo do zaključka, da smo performačno omejeni le s proračunom oziroma performanci strojne opreme. Spodnji diagram prikazuje eno od možnih postavitev sistema, ki vsebuje tri strežnike in razporejevalnik zahtev.



Slika 5.3: Sistem treh centralnih strežnikov

Možna je tudi ločena postavitev poljubnega števila podatkovnih strežnikov in ločena postavitev poljubnega števila spletnih strežnikov. Na ta način dosežemo visoko razpoložljivost

in stabilnost sistema, saj sistem deluje učinkovito tudi v primeru, ko katerikoli strežnik odpove.

5.3 Varnost

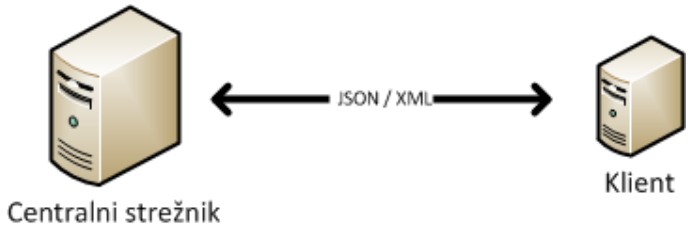
Za varnost je poskrbljeno na več nivojih. Vsi podatki se prenašajo preko varne HTTPS povezave. Vsa gesla in ključi so hranjena v zgoščeni obliki tako, da je ob morebitni kraji podatkov nemogoče dobiti izvorno obliko gesel in ključev. Drugi nivo varnosti je prijava. Vsak klient, tako uporabnik kot strežnik, se mora pred uporabo sistema avtenticirati. S tem preprečimo pošiljanje lažnih podatkov in ogled le teh nepooblaščenim osebam.

5.4 Razširljivost

Sistem je možno poljubno razširiti z izdelavo novih komponent. Za prihodnjo različico sistema je predvidena izdelava komponente za samodejno obveščanje uporabnikov o napakah na sistemu in o pomanjkanju sredstev. To je možno z direktnim pooblaščenim dostopom do baze ali pa kot nadgradnja obstoječega sistema. K enostavnosti nadgradnje pripomore tudi uporabljen princip KISS [25], ki v dobesednem prevodu pomeni "Ohrani enostavnost in neumnost". Kar lahko razumemo na način, da kodo oziroma arhitekturo poenostavimo in ji namenimo le njej namenjeno funkcionalnost in ne da ena magična komponenta skuša rešiti vse težave. Prav tako je možno sistem prilagoditi tako, da se strežniški del poganja na Windows operacijskem sistemu, ali pa tako, da izdelamo klienta za trenutno nepodprt operacijski sistem.

5.5 Izmenjava podatkov

Za izmenjavo podatkov med sistemi sta v splošnem aktualni dve obliki, XML in JSON. Prva se uporablja predvsem v Java svetu ter svetu spletnih storitev SOAP. JSON se pa uporablja predvsem v spletnih tehnologijah in lahkih spletnih storitvah.



Slika 5.4: Izmenjava podatkov

Za implementacijo spletnih storitev se uporabljata predvsem dva tipa. Bolj znane so SOAP spletne storitve. V zadnjem času so vedno bolj popularne REST spletne storitve.

5.6 Uporabniški vmesnik

Spletni grafični vmesnik bo vmesnik med centralnim strežnikom in uporabnikom. Preko njega bo možno opraviti vsa zahtevana opravila. Temeljil bo na standardnih spletnih tehnologijah HTML, CSS in JavaScript. Za boljšo interakcijo z uporabnikom se bo uporabilo eno od JavaScript ogrodij. Komunikacija z bazo bo potekala preko JSON oblike, ki je integrirana v bazo podatkov.

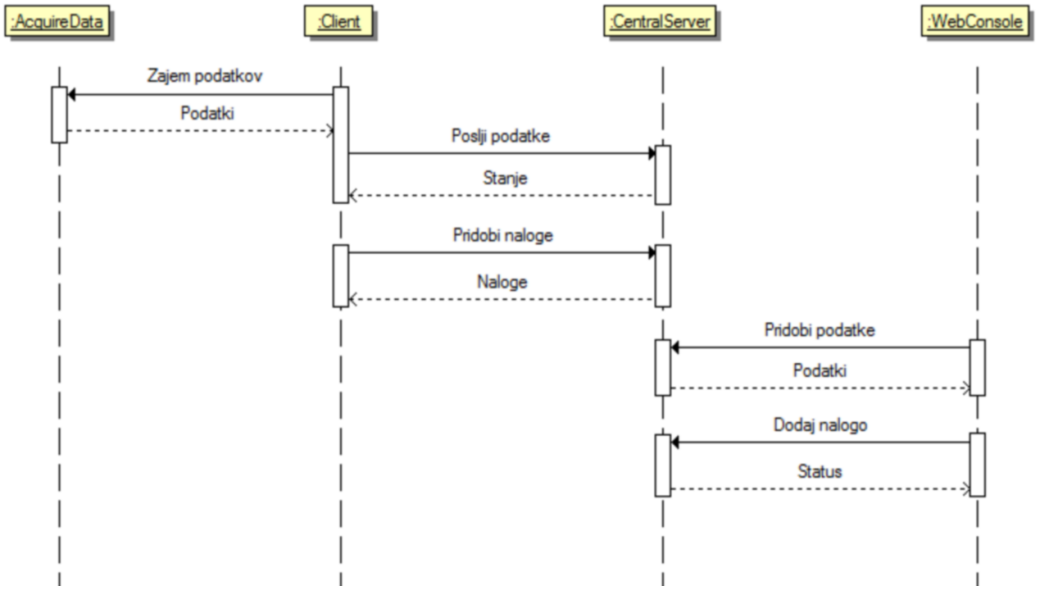
Poglavje 6

Načrtovanje programa

Ker gre za akademski projekt brez dodeljenih finančnih sredstev, se za implementacijo uporabljajo izključno zastonjska, odprtokodna orodja in programska oprema. V tem primeru je na voljo več primernih programskih orodij, zato je pred samo izbiro orodij potrebno dobro premisliti katera uporabiti. Pri izbiri so eni ključnih parametrov dokumentacija, podpora, razširjenost uporabe, prisotnost na trgu in cena. So pa lahko vsi tej parametri tudi varljivi, zato je dobro programska orodja pred samo izbiro tudi dejansko testirati, da na primerih vidimo kaj so prednosti in slabosti. Ugotovitev neustreznosti v pozni fazi projekta lahko pomeni velika časovne zamike, povišane stroške in celo neuspešnost projekta.

6.1 Program

Komponente sistema med seboj komunicirajo preko REST spletnih storitev. Sledeči sekvenčni diagram prikazuje izvajanje aktivnosti.

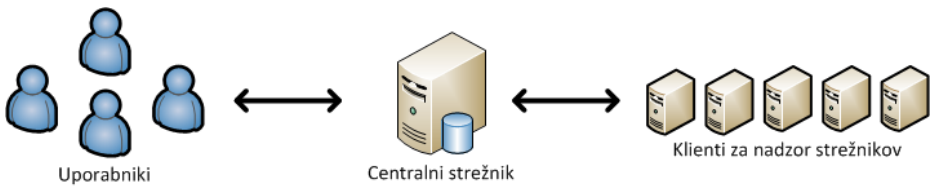


Slika 6.1: Sekvečni diagram sistema

Centralni nadzorni strežnik je edina komponenta sistema, ki neposredno dostopa do podatkovne baze. Ostale koponente dostopajo do podatkov preko spletnih storitev centralnega strežnika. Komunikacija med klientom nadzorovanega strežnika in centralnik strežnikom se izvaja periodično na zahtevo klienta. Klient tako na določen časovni interval pošlje podatke in pridobi opravika, ki so mu dodeljena.

6.2 Centralni strežnik

Centralni strežnik je osrednja komponenta sistema. Z njim komunicirajo vse ostale komponente. Skrbi za prejemanje, pošiljanje in hranjenje podatkov.



Slika 6.2: Strežnik kot osrednji del sistema

Centralni strežnik je edina komponenta sistema, ki neposredno dostopa do podatkovne baze. Ostale komponente dostopajo do spletnih storitev centralnega strežnika, ki nato podatke prebere oziroma zapiše v bazo.

Spletne storitve uporabljajo REST spletne storitve, ki temeljijo na HTTP protokolu. Za uspešen zapis podatkov na server in branje z njega tako potrebujemo tri ključne stvari - uporabniški račun, pravilen spletni naslov ter pravilno HTTP metodo.

Interno lahko centralni strežnik razdelimo na API del za delo s podatki, na strežnik spletnega vmesnika, ter na skupni del obeh prej omenjenih komponent, to je dostop do baze. Spletne storitve za upravljanje podatkov se nahajajo znotraj **/api/*** imenskega prostora.

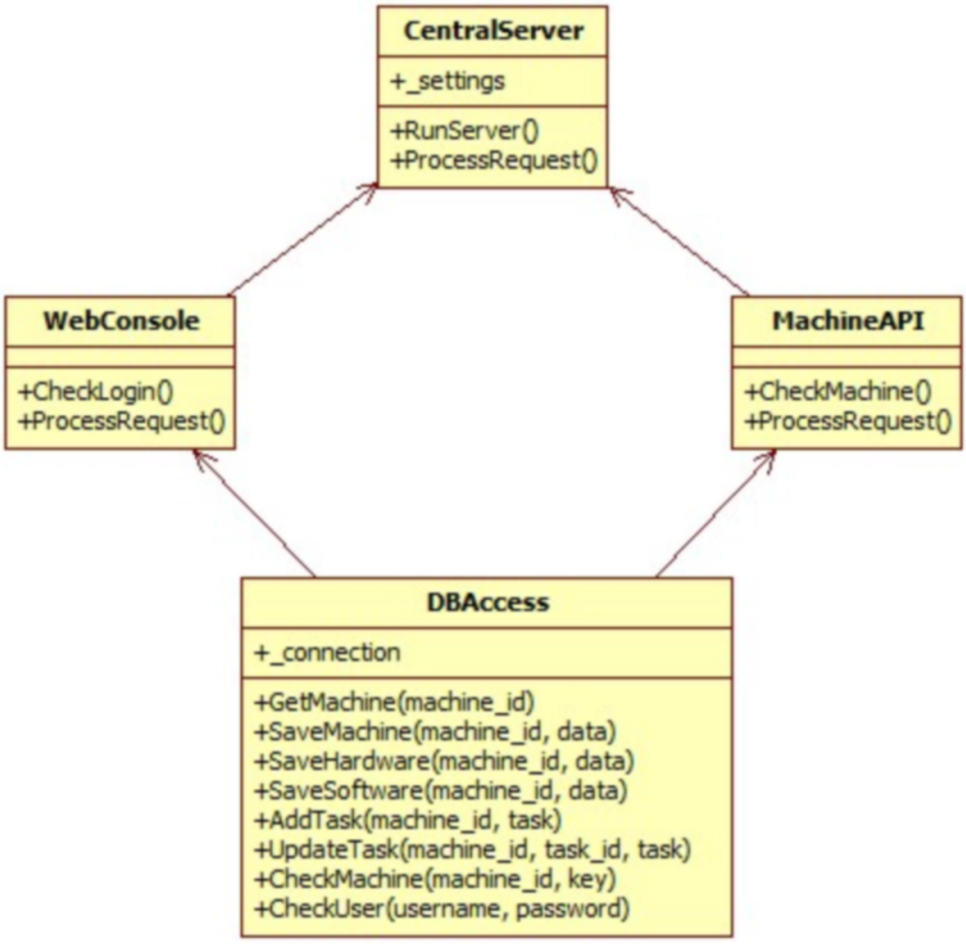
Tabela 6.1: API centralnega strežnika

Metoda	URL	Opis
POST	/api/hardware	Zapis podatkov o strojni opremi
POST	/api/software	Zapis podatkov o programski opremi
POST	/api/processes	Zapis podatkov o procesih
GET	/api/tasks	Pridobitev opravil
POST	/api/taskstatus	Zapis rezultata opravila

Aktivnosti ob novem zahtevku si sledijo v sledečem zaporedju:

- avtentikacija
- validacija in obdelava podatkov
- vrnitev rezultata

Spodnji diagram prikazuje razredni diagram jedra centralnega strežnika.

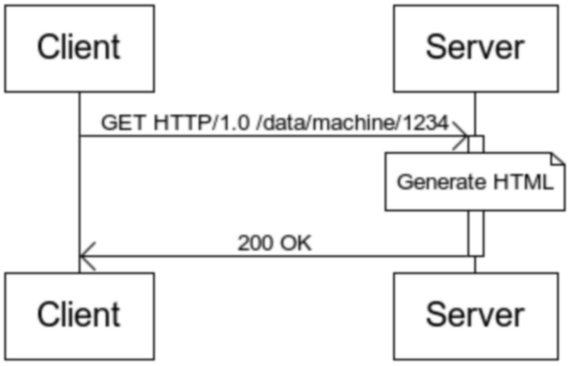


Slika 6.3: UML razredni diagram jedra centralnega strežnika

Vsak zahtevek pride v metodo ProcessRequest, ki na podlagi URLja določi razred, ki bo obdelal zahtevek. Tako vsi zahtevki, ki se pričnejo z URL naslovom /api/ preidejo v ProcessRequest metodo razreda MachineAPI, ki najprej preveri uspešnost avtentikacije, nato preveri tip zahteve. Če je zahteva tipa GET prebere zahtevane podatke iz baze in jih vrne v JSON obliki. Če je zahteva tipa POST najprej preveru ustreznost podatkov, jih zapiše v bazo in vrne uspešnost zapisa.

Vse ostale zahteveke katerih URL se ne prične z /api/ preidejo v ProcessRequest metodo razreda WebConsole, ki najprej preveri uspešnost prijave, ki je shranjena v seji. Če prijava ni uspešna vrne uporabniku informacijo o zavrnjenem dostopu. Če je prijava uspešna se na podlagi tipa zahtevka določi nadaljni potek izvajanja. Če je tipa GET, se zahtevani podatki preberejo iz baze in vrnejo v HTML obliki uporabniku. Če je tipa POST se ustvari novo opravilo, ki se doda na seznam opravil za izbran nadzorovan strežnil. Uporabniku se vrne seznam opravil v HTML obliki, ki jo zgenerira spletni vmesnik.

Centralni strežnik pa hkrati gostuje tudi spletni vmesnik. Po potrebi se lahko spletni vmesnik loči. Spodnji diagram prikazuje obdelavo zahteve spletnega vmesnika, ki izriše zavihek za izbran nadzorovan strežnik.



Slika 6.4: Obdelava zahteve spletnega vmesnika

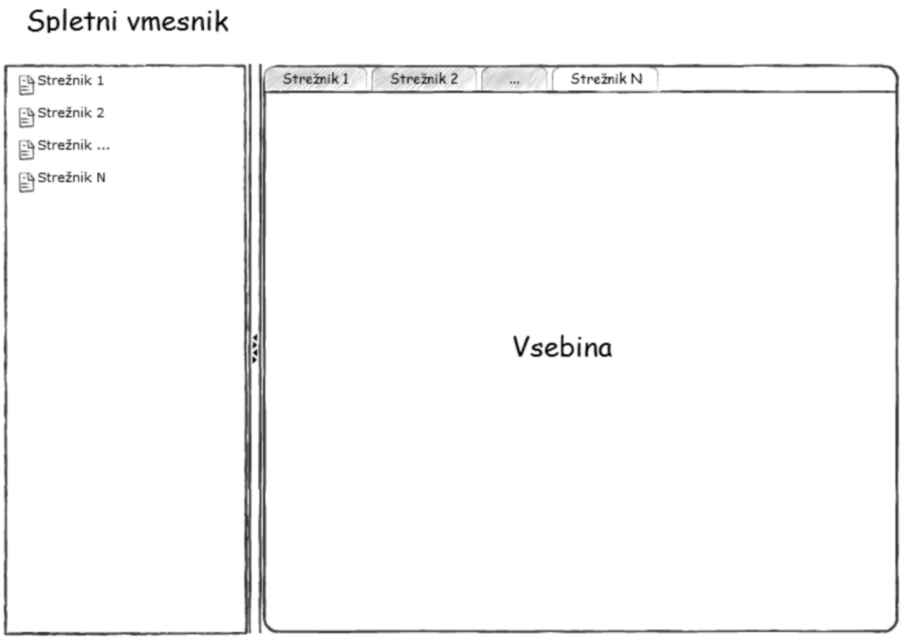
Strežnik kot tak mora zagotavljati visoko razpoložljivost in stabilnost delovanja. To dosežemo tako, da imamo vzporedno pognanih več strežnikov, med katere se razporeja zahteve. Strežniki so kopije, tako kar se programske opreme tiče kot podatkov. Strežniški del lahko ločimo na spletni strežnik in podatkovni strežnik. Glede na potrebe izberemo ustrezno število strežnikov, da bodo ustregli vsem zahtevam.

6.3 Spletni vmesnik

Spletni vmesnik je središče za nadzor in upravljanje registiranih spletnih strežnikov. Je ključni del sistema, saj z njim upravljajo uporabniki. Preko njega lahko opravljamo razne operacije, kot so ponovni zagon, ustavitev strežnika, pregled in ubijanje zagnanih procesov, ustvarjanje skript za izvajanje na klientih in pregled strojne ter programske opreme.

Zaradi varnosti se moramo pred uporabo v sistem avtenticirati z uporabniškim imenom in geslom. Po uspešni prijavi se uporabniku prikaže vmesnik za nadzor in upravljanje. Za nadzor in upavljanje strežnika je najprej potrebno izbrati željen strežnik. Ko izberemo željen strežnik se nam v vsebinskem delu odpre nov zavihek z imenom izbranega strežnika. Vsak zavihek s strežnikom vsebuje šest podzavihkov. Zavihki predstavljajo sklope možnih operacij.

Spletni vmesnik bo izdelan z uporabo standardnih spletnih tehnologij HTML, CSS in JavaScript. Za lažjo izdelavo smo grafičnega vmesnika smo izbrali JavaScript ogrodje Dojo, ki nudi pester izbor gradnikov. Spodnja slika predstavlja skico kako naj bi izgledal spletni vmensik.



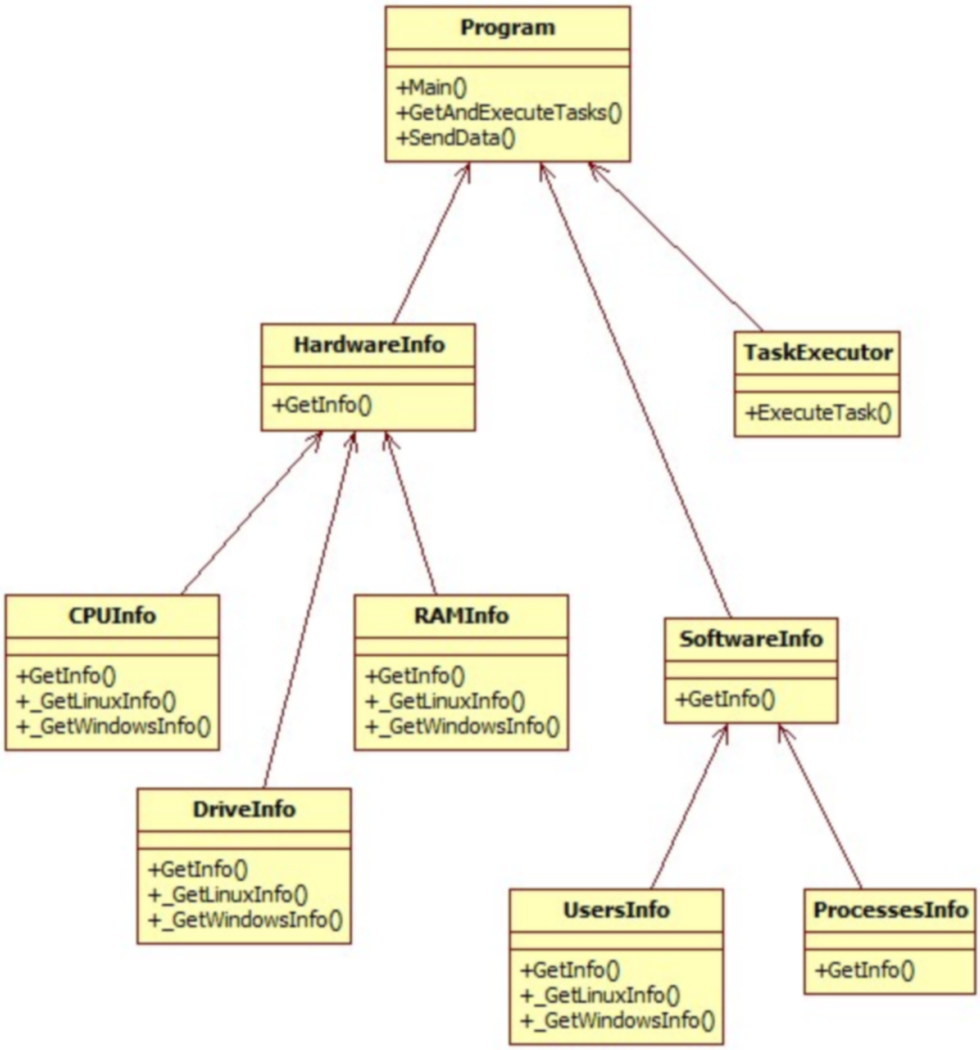
Slika 6.5: Skica spletnega vmesnika

Levo se nahaja seznam strežnikov. Ob kliku na strežnik se nam desno odpre nov zavihek z imenom strežnika. Na ta način imamo hkrati odprtih več zavihkov, vsak prikazuje drug strežnik. S ponovnim klikom na strežnik na seznamu levo se nam zavihek osveži.

6.4 Klient nadzorovanega strežnika

Klient je aplikacija nameščena na nadzorovan strežnik, ki dostopa do centralnega nadzornega strežnika. V našem primeru klient pošilja stanje o resourcih na centralni strežnik, ter preverja ali je zanj nastavljeno kako opravilo. Aplikacija periodtično (čas je nastavljih) pošilja podatke, ki jih strežnik nato obdela.

Zaradi razlik med operacijskimi sistemi je potrebno podatkovne razrede izdelati dovolj abstraktno, da ustrezajo vsem zahtevanim operacijskim sistemov. Datotočeni sistem operacijskega sistema Microsoft Windows se naprimer močno razlikuje od datotečnih sistemov Unix operacijskih sistemov, prav tako pa je način pridobivanja teh podatkov različen.



Slika 6.6: UML razredni diagram klienta nadzorovanega strežnika

Za pridobivanje podatkov se uporabijo funkcije standardne Python knjižice. Ker to ni vedno mogoče, se bo uporabila tudi knjižica Psutil [23], knjižica WMI [24] ter standardna orodja Linux operacijskega sistema. Zaradi lažjega obvladovanja podatkov, bomo nad njimi izdelali abstrakcijo, ki poenoti predstavitev podatkov vseh podprtih operacijskih sistemov.

Poleg zajemanja podatkov in pošiljanja le teh, je ena nazanimivejših komponent klienta časovni razporejevalnik. Le ta mora na nastavljene časovne intervale pošiljati in pridobivati podatke iz strežnika. Pirdobljeni podatki niso nikoli starejši od enega dne. Prav tako pomemben del je pa branje nalog iz strežnika in izvedba le teh. Pomembna je časovna pravilnost izvajanja ter vračanje rezultata nazaj na strežnik. V ta namen bomo izdelali več-nitni časovni razporejevalnik, ki funkcije kliče na podan časovni interval.

Za poenostavitev namestitve klienta bomo izdelali pomožno orodje za registracijo klienta na strežnik. Le ta s sklicem na strežnik pridobi nov unikaten ID ter ključ za dostop do

strežnika. Te nastavitve shrani v klientovo nastavitveno datoteko. Uporaba tega orodja je enkratna aktivnost, ki se izvede le ob namestitvi klienta.

6.5 Programska oprema

Dober izbor programske opreme je lahko ključen za uspešno implementacijo sistema. Slaba izbira orodij lahko pomeni težave v kasnejši fazi implementacije. V tej fazi so zelo pomembne tudi izkušnje osebe oziroma skupine oseb, ki izbira orodja. Sledi predstavitev predlaganih orodij, ki jih lahko uporabimo pri implementaciji sistema.

6.5.1 Linux

Linux je prost operacijski sistem s prosto dostopno izvorno kodo in izdan pod licenco GPL. Zaradi svoje odprtosti in brezplačnosti predstavlja ustrezno izbiro za strežniški del našega sistema. Linux se odlično izkaže pri poganjanju spletnih aplikacij in storitev.

Za naše potrebe smo izbrali Ubuntu distribucijo operacijskega sistema Linux, ki pripada družini Debian distribucij. Konkretno smo uporabili Ubuntu Server inačico, ki je posebej namenjena spletnim strežnikom in ne vsebuje nepotrebnega grafičnega vmesnika. Ubuntu je trenutno najpopularnejša distribucija Linux operacijskega sistema, saj je ena prvih, katerih cilj je enostavna uporaba in dobra podpora strojne opreme. Možno je tudi dobiti poslovno podporo s strani podjetja, ki stoji za razvojem Ubuntuja, podjetja Cannonical Ltd.

6.5.2 Python in Bottle

Python programski jezik je izbran zaradi tega, ker je implementiran za vse popularne operacijske sisteme, vključujoč Windows, Linux, FreeBSD, OS X in ostale. Delo z njim je enostavno, dokumentacija odlična, pestrost knjižic pa neskončna. Zaradi vseh teh lastnosti je primeren tako za strežniški del kot tudi za kliente.

Bottle odprtokodno micro spletno ogroje za programski jezik Python [20]. Spletno ogrodje je skupek orodij in funkcij, ki olajšajo razvoj spletni strani in spletnih aplikacij. Ogrodje v svetu programske opreme predstavlja povezan skupek funkcij in orodij, ki so pogosto uporabljene za rešitev podobnih problemov. Z izdelavo ogrodij se odpravi ponavljajoča se opravila, standardizira rešitve in skrajša razvoj programske opreme. Bottle spletno ogrodje vključuje podporo za preusmerjanje, predloge, razna pripomočke in strežniško podporo za izvajanje aplikacije. Ena od značilnosti Bottle ogrodja je, da je v celoti zapisan v eni datoteki imenovani bottle.py. Prednost tega je, da je sam prenos in postavitev enostavnejša, saj ga lahko enostavno priložimo sami aplikaciji in se tako izognemu potrebi po nameščanju dodatnih paketkov na ciljnem strežniku. Primer enostavne spletne strani izdelane z uporabo Bottle:

```
#!/usr/bin/env python
from bottle import route, run

@route('/hello/:name')
def index(name='World'):
```

```
    return '<b>Hello %s!</b>' % name

run(host='localhost', port=8080)
```

Aplikacijo poženemo z ukazom ./bottle.py, ogledamo pa si jo lahko na spletnem naslovu http://localhost:8080/hello/FAMNIT. Privzeto bottle uporabi refrenčno implementacijo spletnega strežnika, ki je del Python standardne knjižice, imenovano wsgiref. Za naš sistem uporabljamo drugi spletni strežnik. Po testiranju pritiska na prototipu smo ugotovili, da se najboljše obnese Tornadoweb spletni strežnik.

6.5.3 MongoDB

Predstavljen sistem vključuje množico podatkov, ki jih je potrebno nekako hraniti. V ta namen se uporabi baze podatkov, ki skrbijo, da se podatki varno shranjeni. Obstaja veliko podatkovnih baz, ki zadovoljujejo različne potrebe. Ena popularnejših odprtokodnih in zastonjskih podatkovnih baz je MySQL. To je relacijska podatkovna baza, ki zadovoljejo potrebe večine spletnih strani in aplikacij. Eden od problemov MySQL podatkovne baze, tako kot ostalih relacijskih podatkovnih baz je to, da mora biti shema podatkov dobro definirana. Kasnejše spremembe na shemi so lahko težavne, v skrajnih primerih celo nemogoče. Ker je vodilo našega informacijskega sistema enostavnost in razširljivost, smo uporabili eno od v zadnjem času zelo popularnih, takoimenovanih No-SQL podatkovnih baz. Za razliko od relacijskih podatkovnih baz te baze nimajo dobro definirane sheme podatkov. Podatki so raje predstavljeni kot poljubni dokumenti. V ožji izbor sta tako prišli dve podatkovni bazi - Apache CouchDB in MongoDB [21]. Obe izpolnjujeta zahtevo po hitrosti operacij in varnosti shranjenih podatkov, zato je na izbiro vplivala boljša podpora MongoDB v Python programskem jeziku. Ena od prednosti podatkovne baze MongoDB je tudi ta, da že privzeto uporablja UTF-8 kodiranje znakov.

MongoDB je odprtokodna, visoko-zmogljivostna, sheme-svobodna, dokumentno orientirana podatkovna baza, napisana v programskem jeziku C++. Poganja jo lahko operacijski sitemi Windows, Linux, OS X in Solaris. Podatki so predstavljeni v JSON podobni obliki. Podporo ima praktično v vseh programskih jezikih. MongoDB uporablja BSON [**?**], to je binarni zapis JSON.

MongoDB omgoča poizvedovanje po poljubnem polju ob poljubnem času, razponsko poizvedovanje, pozivedovanje z uporabo regularnih izrazov in ostale poizvedbe. Poizvedbe lahko vključujejo uporabniško definirane JavaScript funkcije - če funkcija vrne true je dokument vključen, v nasprotnem primeru ni. Primer dokumenta v MongoDB:

```
{
    "uporabnik" : "Andrej",
    "naslov" : {
        "ulica" : "Frenkova cesta",
        "hisna_stevilka": 24,
        "postna_stevilka" : 6276,
        "drzava": "Slovenija"
    }
}
```

Na takem dokumentu lahko izvedemo poizvedbo, ki išče po določenem polju v seznamu lastnosti. V primeru, da želimo v bazi uporabnikov najti vse uporabnike, kateri živijo na poštni številki 6276 lahko izvedemo naslednjo poizvedbo:

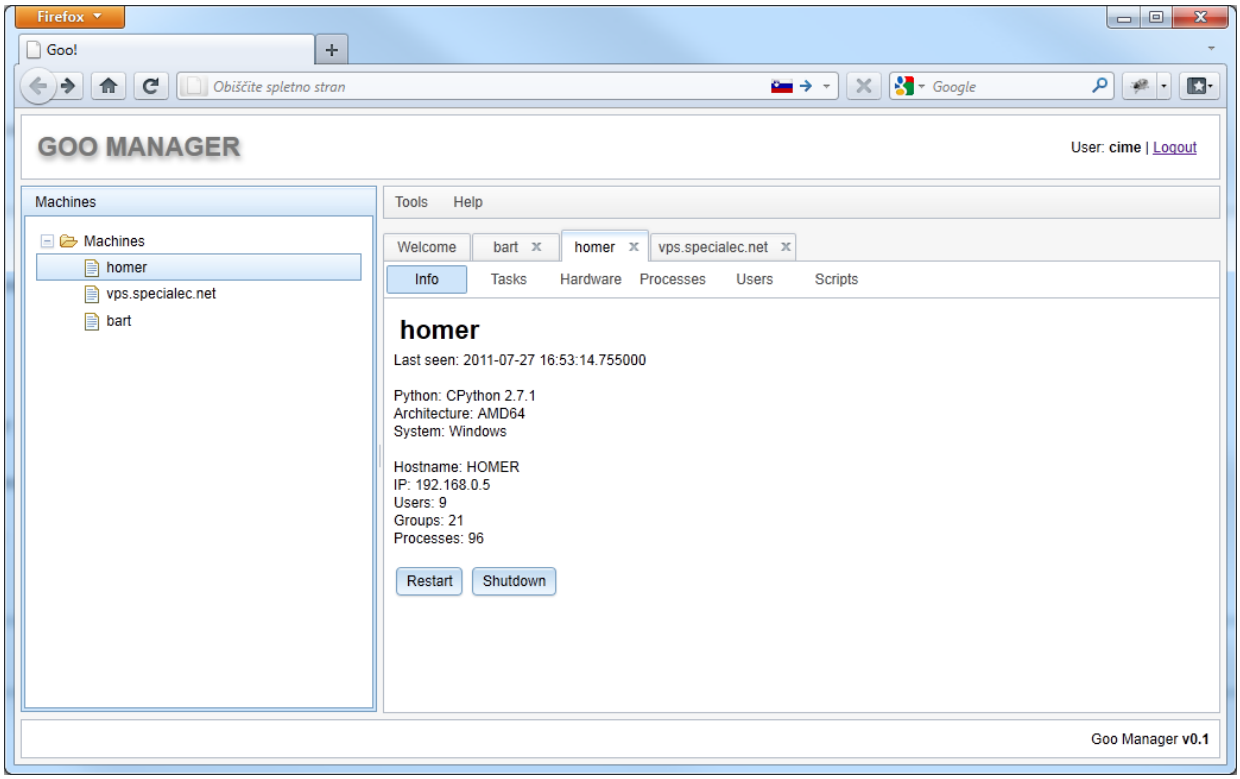
```
db.uporabniki.find({"naslov.postna_stevilka" : 6276})
```

V primeru, da iskano polje ne obstaja poizvedba ne vrne rezultatov.

6.5.4 Dojo

Ker izdelujemo spletno aplikacijo je potrebno izdelati tudi spletni vmesnik, ki je prijazen do uporabnika. Za izdelavo spletnih vmesnikov in spletnih strani se navadno uporablja HTML označevalni jezik, CSS prekrivni slogi in JavaScript programski jezik. Obstaja več ogrodij za spletne gradnike, mi pa bomo uporabili Dojo, ki temelji na standarnem HTML, CSS in JavaScript.

Dojo, nepremagljivi JavaScript, vsebuje mnoge spletne gradnike kot so drevo, tabela, zavihki, grafi, gumbi, koldar in orodne vrstice. Prav tako pa vsebuje močen upravljalnik za razporejanje elementov, s katerim lahko ločimo vmesnik na posamezne dele, ki jim je mogoče poljubno spreminjati velikost. Na ta način smo naš vmesnik razedlili na štiri glavne dele - na glavo, nogo, levo stransko vrstico in vsebinski del.



Slika 6.7: Spletni vmesnik

Razmerje velikosti med stransko vrstico in vsebinskim delom je možno poljubno določati, kar nam omogoča v Dojo vgrajena funkcionalnost.

Dojo se vsebinsko deli na tri sklope. Prvi, imenovan Dojo, predstavlja srce ogrodja in razširja in standardizira osnovno funkcionalnost JavaScripta, kar vključuje poizvedbe, animacije, podporo potegni in spusti in AJAX. Drugi, nam najzanimivejši sklop, predstavljajo gradniki za grafični vmesnik. Ta vsebuje raznorazne komponente, vidne tudi v standardnih namiznih aplikacijah. Uporaba le teh močno pohitri razvoj, hkrati pa ponuja poznane kontrole in prijazen vmesnik. Tretji sklop predstavljajo razvojno vejo Dojota, katera vsebuje Dojo in Dijit komponente, ki še niso čisto pripravljene na produkcijo oziroma ne spadajo ne v Dojo ne v Dijit sklop.

6.5.5 REST

Representational State Transfer (REST) je oblika programske arhitekture za distribucijo podatkov na sistemih kot je splet. Predstavil ga je Roy Fielding v svoji doktorski dezertaciji leta 2000, ki je tudi eden od prvotnih avtorjev protokola HTTP. REST je lahkotna alternativa standardu SOAP, ki za prenos podatkov uporablja XML. Za prenos REST uporablja standardni protokol HTTP, tako da uporablja že definirane funkcije GET, POST, PUT, DELETE. Spletne strani navadno uporabljajo samo funkcije GET in POST. Za razliko od SOAP, REST uporablja že definirane funkcije protokola HTTP, kot je predpomnjenje, avtentikacija in pogajanje za vrsto vsebine, ter se tako izogiba ponovnemu izumljanju rešitev za znane probleme.

Ključni cilji REST so:

- nadgradljivost
- posplošenje vmesnikov
- neodvisna postavitev komponent

RESTful spletne storitve so enostavne spletne storitve implementirane z uporabno HTTP protokola in REST principi. Definirane so v treh pogledih:

- spletni URL naslov, naprimer http://spletna.aplikacija.com/api/
- oblika prenosnega medija, kot sta JSON in XML
- set HTTP operacij - GET, POST, PUT, DELETE

HTTP operacije so navadno uporabljene na ta način:

- GET - pridobi podatek
- POST - posodobi podatek
- PUT - ustvari podatek
- DELETE - izbriše podatek

Poglavje 7

Izvedba

V tem poglavju je predstavljena praktična implementacija informacijskega sistema za centraliziran nadzor strežnikov. Uporabljena orodja so odprtokodna in prostodostopna. Predstavljeno je tudi delovanje programa na samih primerih uporabe.

7.1 Uvod

Izvedba je ena najzahtevnejših in najkritičnejših faz, saj se pri njej najbolj opazijo vse morebitne napake napravljene v prejšnjih fazah. Slaba analiza problema in zahtev lahko pomeni propad projekta. Pomembno je, da so v predhodnih fazah podane realne ocene in pričakovanja. Slaba definicija zahtev v povezavi s slabo oceno potrebnih sredstev lahko pomeni v najslabšem primeru tudi propad podjetja.

7.2 Centralni nadzorni strežnik

Strežniški del je osrednji del sistema. Do njega dostopajo klienti in spletni vmesnik. Skrbi za hrambo in obdelavo podatkov, zato dostopa direktno do podatkovne baze MongoDB, preko knjižice PyMongo, ki je uradna knjižica MongoDB za Python. Poganja ga spletni ogrodje za izdevo spletnih strani v Pythonu imenovano Bottle. Glavna naloga je, da podatke na zahtevo bere in piše v bazo. Podatke si s klientom in spletnim vmesnikom izmenjuje preko lahkih spletni storitev REST, temlječih na protkolu HTTP.

Največji izziv pri implementaciji te komponente je predstavljala izdelava hitrega in odzivnega spletnega spletnega strežnika, ki bo podatke pametno shranjeval, da bodo dostopi do njih karseda hitri.

Strežniški del vsebuje nastavitveno datoteko, katera hrani podatke za dostop do podatkovne baze ter podatke o strežniku samem. Datoteka je sledeče oblike:

```
;Goo server
server_address = 0.0.0.0
server_port = 4321

;MongoDB
```

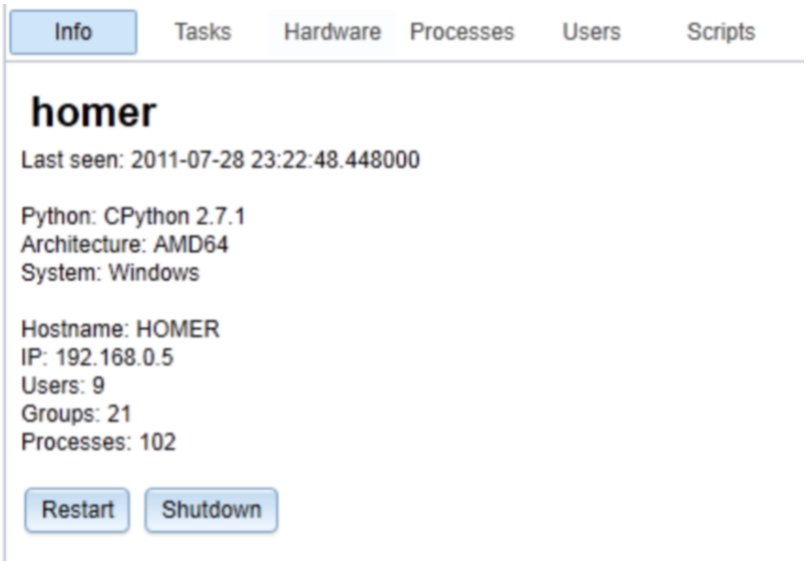
```
mongodb_replicaset = True
mongodb_server = localhost
mongodb_database = goo
```

7.3 Spletni vmesnik

Spletni vmesnik je središče za nadzor in upravljanje registiranih spletnih strežnikov. Je ključni del sistema, saj z njim upravljajo uporabniki. Preko njega lahko opravljamo razne operacije, kot so ponovni zagon, ustavitev strežnika, pregled in ubijanje zagnanih procesov, ustvarjanje skript za izvajanje na klientih in pregled strojne ter programske opreme.

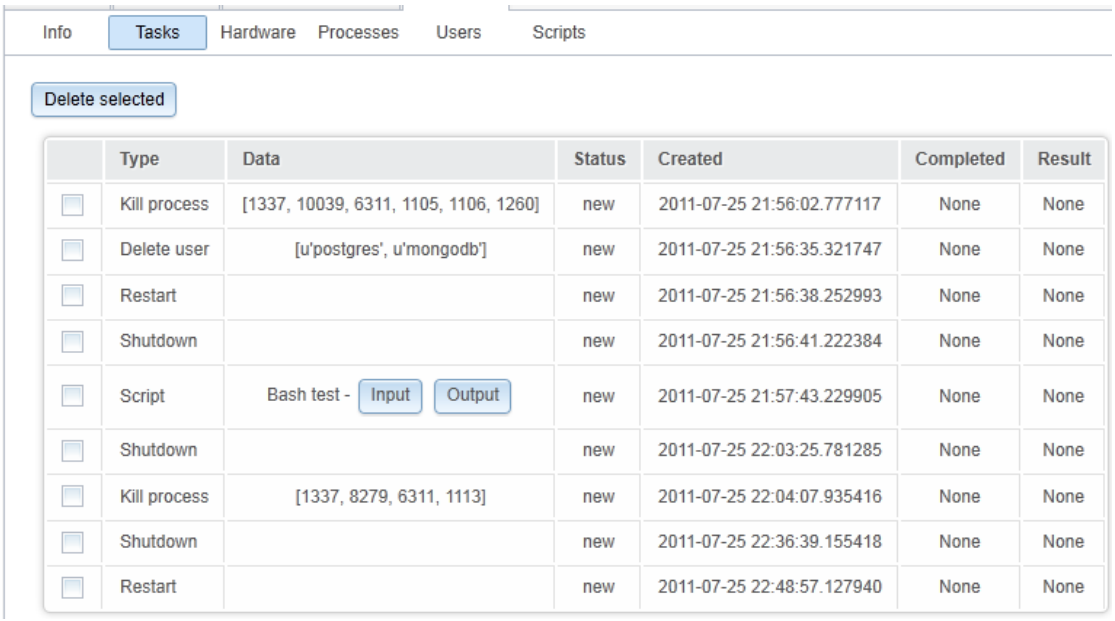
Največji izziv je bila izdelava sodobnega, odzivnega, uporabniku prijaznega spletnega vmesnika. Uporabili smo standardne spletne tehnologije kot so HTML, CSS in JavaScript (vključno z Ajax-om). Sam vmesnik se povezuje na podatkovno bazo, ki je skupna s centralnim nadzornim strežnikom.

Pred uporabo se moramo v sistem avtenticirati z uporabniškim imenom in geslom. Ko se uspešno prijavimo, se nam odpre uporabniški vmesnik za upravljanje, ki na začetku prikazuje le pozdravno vsebino. Za nadzor in upavljanje strežnika moramo najprej izbrati željen strežnik v stranski vrstici, v drevesnem korenu Machines. Ko kliknemo na željen strežnik se nam v vsebinskem delu odpre nov zavihek z imenom izbranega strežnika. Vsak zavihek s strežnikom vsebuje 6 podzavihkov. Privzeto je izbran zavihek Info, ki prikazuje osnovne podatke strežnika, ter gumba za povnovni zagon in zaustavitev.



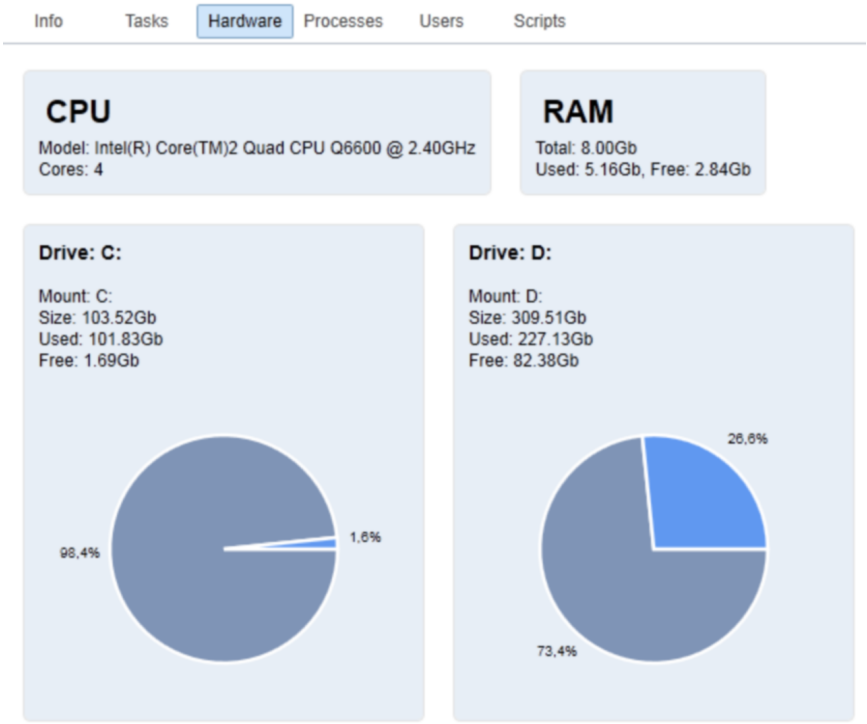
Slika 7.1: Osnovne informacije nadzorovanega strežnika

Sledi mu zavihek Tasks na katerem so prikazane naloge za strežnik. Na seznamu vidimo, kdaj je bila naloga dodana, kašno je njeno stanje in rezultat.



Slika 7.2: Seznam nalog nadzorovanega strežnika

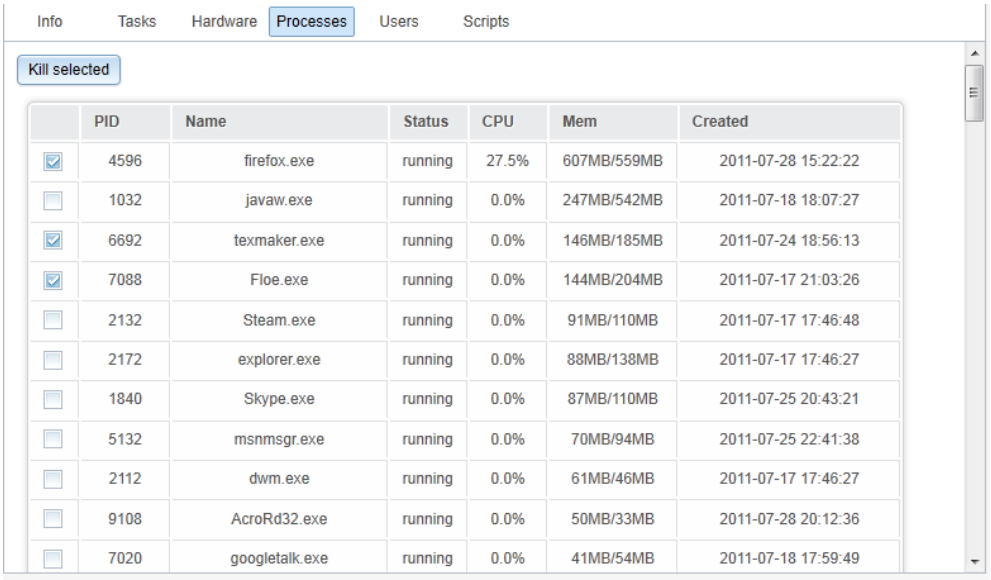
Nato sledi zavihek Hardware, ki prikazuje informacije o strojni opremi, kot so informacije o centralnem procesorju, o količini in zasedenosti pomnilnika, ter informacije o diskih.



Slika 7.3: Podatki o strojni opremi

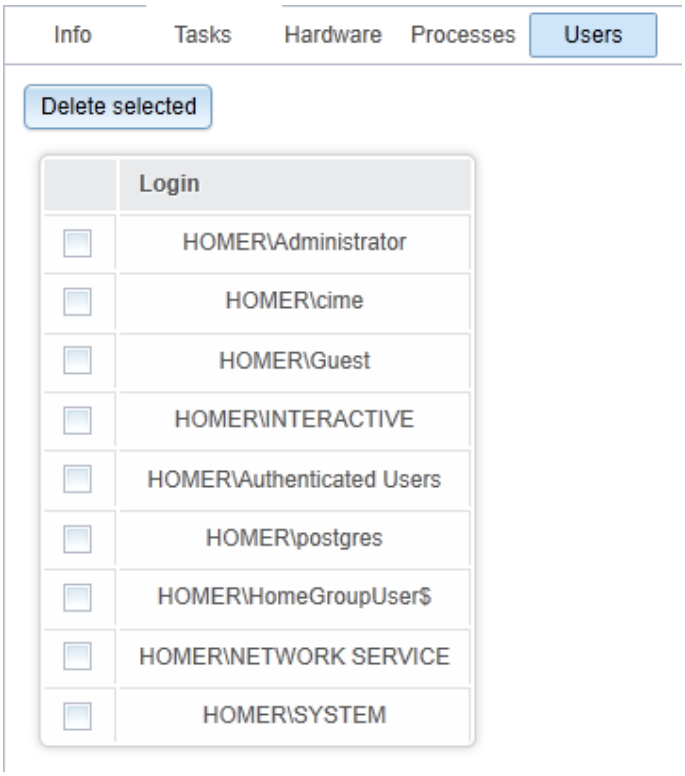
Na zavihku Processes vidim vse tekoče procese. Če želimo katerega izmed njih ubiti,

ga obkljukamo in kliknemo gumb Kill selected, ki na seznam nalog doda nalogo za ubitje izbranih procesov.



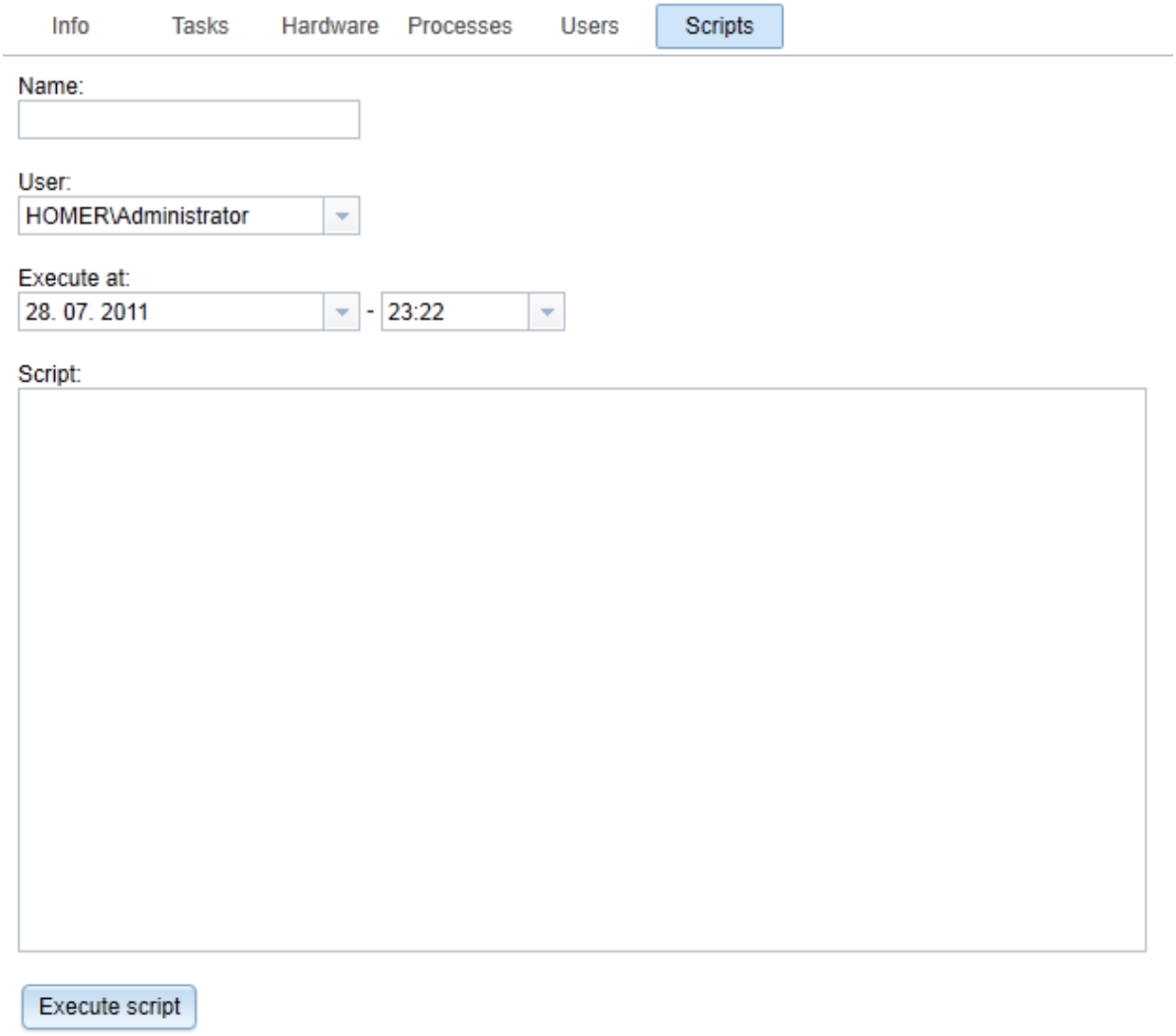
Slika 7.4: Seznam zagnanih procesov

Na zavihku Users vidimo seznam uporabnikov izbranega strežnika.



Slika 7.5: Seznam uporabnikov

Zadnji zavihek je Scripts, ki je najmočnejša funkcionalnost sistema. Preko njega lahko na strežniku storimo praktično karkoli.



Slika 7.6: Izdelava sistemske skripte

7.4 Klient nadzorovanega strežnika

Klient je aplikacija, ki dostopa do strežnika. V našem primeru klient pošilja stanje o resourcih na centralni strežnik, ter preverja ali je zanj nastavljeno kako opravilo. Aplikacija periodtično (čas je nastavljih) pošilja podatke, ki jih strežnik nato obdela.

Program zajema podatke na podlagi operacijskega sistema na katerem je pognan. V Pythonu v ta namen uporabimo standardno knjižico platform:

```
import platform
```

```
platform.system()
```

Podatki se zajemajo z uporabo različnih knjižic in sistemskih pripomočkov. Tako se za procese uporablja knjižica Psutil, ki deluje tako na Windows kot Linux operacijskih sistemih. Za zajem podatkov o stanju diskov se na Windows operacijskem sistemu uporablja knjižica WMI, na Linuxu pa kar sistemski pripomoček dh, ki v tekstovni obliki vrne stanje vseh diskov na sistemu.

Za izvajanje skript je implementirana le osnovna funkcionalnost. Tako skripto izvede uporabnik, ki poganja klienta, v večini primerov root uporabnik oziroma administrator sistema.

Poleg samega zajema podatkov program opravlja še dve pomembni nalogi. Prva je pošiljanje in prejemanje podatkov iz strežnika. To nalogo opravlja periodično, na nastavljen interval. Druga naloga pe je branje opravil iz strežnika, kot so ubitje procesov in ponovni zagon sistema. Ko program nalogo prebere jo ob podanem času izvede in rezultat vrne nazaj na strežnik.

Klient je implementiran v programskem jeziku Python, z uporabo knjižic WMI in Psutil.

Klient vsebuje tudi nastavitveno datoteko. Datoteka je sledeče oblike:

```
;Goo server
server = 192.168.0.2:4321

;Machine
machine_id = 1234
machine_key = 12344321
```

7.4.1 Pripomoček za namestitev

Za poenostavitev namestitve klienta smo izdelali pomožno orodje za registracijo klienta na strežnik. Le ta s sklicem na strežnik pridobi nov unikaten ID ter ključ za dostop do strežnika. Te nastavitve shrani v klientovo nastavitveno datoteko. Uporaba tega orodja je enkratna aktivnost, ki se izvede le ob namestitvi.

Tako kot klient in strežnik je tudi ta pripomoček implementiran v programskem jeziku Python. Predstavlja najenostavnejšo komponento našega sistema.

```
# python goo-configure
Enter machine name:
Machine successfuly registered with server!
```

Poglavje 8

Testiranje

Testiranje je bilo izvedeno v virtualiziranem okolju z osmimi virtualnimi strežniki. Vseh osem strežnikov je sočasno pošiljalo in prejemalo podatke iz centralnega strežnika. Cilj testiranje je bil preveriti delovanje vseh komponent sistema. Testerji so navadno razvojni inženirji, ki dobro poznajo sistem, jim je glavni interes pravočasna zaključitev projekta in so hkrati kontstruktivni, ter neodvisni preizkuševalci, ki sistema ne poznajo in so destruktivni. Slednji odkrijejo največ napak v sistemu. Zaradi omejenih sredstev sistem ni bil temeljito stestiran s strani neodivsne osebe oziroma oseb. Posamezne komponente so bile testirane že med samo implementacijo. Po končani implementaciji je sledilo testiranje za uporabnika najpomembnejše komponente, to je spletnega vmesnika sistema.

Pred testiranjem je bila pripravljena množica scenarijev, katere se je preizkusilo na vseh registriranih strežnikih. Vsi testi so bili povezani z zahtevami, saj je izpolnjevanje le teh pogoj, da je lahko projekt uspešen. Ker je nemogoče preizkusiti vse možnosti, smo izbrali tiste za katere smo menili, da so ključne in hkrati obstaja največja možnost napake. Pri pripravi testov smo si pomagali s predhodno opravljeno analizo zahtev in izdelanimi use-case diagrami. Rezultati se nahajajo med prilogami.

Ključne osebe pri testiranju so ciljni uporabniki oziroma naročniki produkta. To so osebe, ki v končni fazi produkt tudi odobrijo in prevzamejo, kar za nas pomeni uspešnen zaključek projekta. Ti testerji so najdestruktivnejši, saj sistem testirajo na vse možne načine, tudi take, ki niso predvideni. Na ta način se odkrijejo napake, ki med sami razvojem niso bile vidne.

8.1 Testiranje spletnega vmesnika

Testiranje spletnega vmesnika je težko avtomatizirati. Potrebno je ročno delo, ki zahteva doslednost in natančnost. Pripravili smo sledeče teste, ki smo jih testirali na različnih strežnikih, med katerimi so bili tako Linux kot Windows strežniki:

- Pregled ujemanja stanja zasedenosti diskov
- Ubitje izbranega procesa
- Zagon sistemske skripte
- Ponovni zagon

8.1.1 Pregled ujemanja stanja zasedenosti diskov

Pri tem testiranju smo testirali ujemanje stanja zasedenosti diskov. To smo testirali tako, da smo sočasno gledali stanje preko spletnega vmesnika in stanje na strežniku samem. Pri Windows klientu ni bilo odkritih napak. Na Linux operacijskem sistemu pa je bila odkrita napaka. Razlog za napako je pa v različnih privzetih količinah, ki jo različne distribucije Linuxa vračajo. Problem je tako bil le pri pretvorbi iz byte-ov v Mega byte, Giga byte in Tera byte. Napaka je bila dodana na seznam napak, ki jih je potrebno odpraviti.

8.1.2 Ubitje izbranega procesa

S tem testiranjem smo ugotavljali delovanje sistema za ubijanje procesov. Test smo opravili tako, da smo na testnih strežnikih pognali naš testni proces in ga preko vmesnika poizkušali ubiti. Konkretno smo pognali spletni brskalnik Firefox in ga ubili. Ker za ubijanje procesov tako Linux kot Windows klient uporabljata isto knjižico, je bil test enako uspešen na vseh strežnikih. Klient v roku minute po dodanem opravilu za ubitje procesa proces tudi ubije. Test je bil uspešen.

8.1.3 Zagon sistemske skripte

Pri tem testu smo pripravili ločeni sistemski skripti za Linux in Windows kliente. Poleg skripte smo pripravili tudi pričakovan rezultat skripte. Testirali smo tako, da smo na izbranih strežnikih skripto pognali in preverili ali se rezultat ujema s pripravljenim. V tem delu smo pričakovali največ težav oziroma napak, saj se tu sistem najbolj razlikuje med posameznimi klienti. Težave so se pojavile pri razlikah v ukazni lupini Linux operacijskega sistema te ukazni vrstici Windows operacijskega sistema. Med testiranjem smo odkrili nepravilnost v delovanju izbire uporabnika, ki skripto poganja. Prav tako smo ugotovili pomanjkljivost klienta v tem, da ne vrne izhoda standardne napake. Te dve napaki oziroma pomanjkljivosti sta bili dodani na seznam težav, ki jih je potrebno odpraviti. V primeru poizkusa zagona skripte za Linux opercaijski sistem na Windows operacijskem sistemu (in obratno) sistem vrne napako, saj se skripta ne more izvesti.

8.1.4 Ponovni zagon

To testiranje je preverjalo delovanje gumbov za ponovni zagon. Ker so se strežniki nahajali na isti fizični lokacijo razdalja ni predstavljala problem. Tako smo osebno lahko spremljali ali se izbran strežnik resnično ponovno zažene. Test je bil uspešen tako na Linux kot Windows klientih.

8.2 Testiranje varnosti

Varnost smo testirali tako, da smo se v sistem skušali prijaviti s pravilnimi in napačnimi uporabniškimi podatki. Najprej smo preizkusili prijavo brez uporabniškega imena in gesla, ter kombinacijami izpuščanja enega od podatkov. Prijava je bila vsakič neuspešna, saj je sistem zavrnil prijavu zaradi neustreznih podatkov. Sledil je vnos pravilnega uporabniškega

imena in napačnega gesla, kar je sistem ponovno prepoznal kot napačno prijavo. Kombinacijo pravilnega uporabniškega imena in pripadajočega gesla je sistem prepoznal kot ustrezno prijavo.

8.3 Testiranje pritiska

S testiranjem pritiska smo želeli preveriti koliko hkratnik zahtev lahko zdrži strežniški del sistema. Ta test smo opravili s preverjenim orodjem Apache ApacheBench [**?**]. Rezultati so pokazali, da lahko testni strežnik brez težav obdela preko 100 sočasnih zahtevkov, kar močno presega zahteve. Performančnost je možno močno izboljšati z nadgradnjo strojne opreme strežnika oziroma zamenjavo le tega. Prav tako je aplikacijo možno nadgraditi tako, da se izvaja na več strežnikih hkrati. Zaradi značilnosti aplikacije je moč zagotoviti zmogljivost tudi za velike podatkovne centre, katere poganjajo stotine strežnikov.

Poglavje 9

Zaključek

Skozi načrtovanje in implementacijo sistema smo na praktičnem primeru videli, kako poteka projekt načrtovanja programske opreme. Videli smo kako pomembna je dobra analiza zahtev za nadaljno načrtovanje in razvoj projekta. Pomembno je, da v tej fazi sodeluje strokovnjak tako na naročnikovi kot izvajalski strani, ki dobro pozna vsebino problema, hkrati pa razlikuje med potrebami in željami. Iz analize zahtev izhaja celoteno načrtovanje in implementacija, ki ji sledi testiranje in validacija. Tako lahko pridemo do zaključka, da je analiza zahtev ena najpomembnejših faz načrtovanja, če ne kar najpomembnejša.

Izvedena implementacija prepušča odprte možnosti za nadgradnje in za različne možnosti postavitve, s katerimi zadovoljimo potrebam naročnika. Tako v najenostavnejši postavitvi zadostuje že en centralni strežnik, ki je sposoben zagotoviti ustrezno delovanje nekaj deset nadzorovanim strežnikom. V primeru porasta števila klientov pa lahko sistem razširimo z dodatnimi centralnimi strežniki, pred katere postavimo usmerjevalnik zahtev, tako da se le te enakomerno razporedijo na vse centralne strežnike.

Ena od zanimiv reči, ki bi jih lahko implementirali v prihodnosti je obveščanje uporabnika o kritičnih stanjih. Ker je sistem modularen in so vsi podatki na voljo v podatkovni bazi, verjamemo da implementacija ne bi bila preveč zahtevna in časovno potratna.

Ob zaključku projekta si želimo, da bo produkt dobro služil naročniku in da bo s tem naše delo opravičeno.

Literatura

[1] Alistair Cockburn, *Writing Effective Use Cases*, (2000).

[2] Mike Cohn, *Agile Estimating and Planning*, (2005).

[3] Tom DeMarco, Timothy Lister, *Peopleware: Productive Projects and Teams*, (1999).

[4] Martin Fowler, *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, (2003).

[5] Erich Gamma, Richard Helm, Ralph Johnson, John M. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, (1994).

[6] Jim Highsmith, *Agile Project Management: Creating Innovative Products*, (2004).

[7] Caig Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*, (2003).

[8] Robert C. Martin, *Agile Software Development, Principles, Patterns, and Practices*, (2002).

[9] Steve McConnell, *Code Complete: A Practical Handbook of Software Construction*, (2004).

[10] Steve McConnell, *Rapid Development*, (2003).

[11] Leonard Richardson, Sam Ruby, David Heinemeier Hansson, *Restful Web Services*, (2007).

[12] Matthew A. Russell, *Dojo: The Definitive Guide*, (2008).

[13] Ken Schwaber, *Agile Project Management with Scrum*, (2004).

[14] Ken Schwaber, Mike Beedle, *Agile Software Development with Scrum*, (2001).

[15] Karl Wiegers, *Software Requirements*, (2003).

Viri

[16] http://islovar.org/, *islovar, Slovar informatike*, (2011).

[17] http://www.canonical.com/enterprise-services/ubuntu-advantage/landscape, *Landscape, an easy-to-use systems management and monitoring service* , (2011).

[18] http://spacewalk.redhat.com/, *Spacewalk, Free & Open Source Linux Systems Management*, Open source, GPLv2 license, (2011).

[19] http://www.nagios.org/, *Nagios - The Industry Standard In IT Infrastructure Monitoring*, Open source, GPL license, (2011).

[20] http://bottlepy.org/, *Bottle - fast, simple and lightweight WSGI micro web-framework for Python.*, Open source, MIT license, (2011).

[21] http://www.mongodb.org/, *MongoDB, scalable, high-performance, document-oriented database*, Open source, GNU AGPL license, (2011).

[22] http://dojotoolkit.org/, *Dojo, Unbeatable JavaScript Tools*, Open source, modified BSD license, (2011).

[23] http://code.google.com/p/psutil/, *psutil, a cross-platform system and process utilities module for Python*, Open source, New BSD license, 2011.

[24] http://timgolden.me.uk/python/wmi/index.html, *WMI, Windows Management Instrumentation* Open source, MIT license, 2011.

[25] http://en.wikipedia.org/wiki/KISS_principle, *KISS principle, Keep it simple, Stupid*

Priloge

- Zgoščenka s celotno izvorno kodo aplikacije.
- Rezultati testiranja vmesnika.
- Rezultati testiranja pritiska.

Testiranje vmesnika

Kaj	Kako	Pričakovan rezultat	Rezultat	Datum
Prijava	Uporabnik vnese napačno uporabniško ime in geslo	Prijava zavrnjena	Prijava zavrnjena	7.9.2011
	Uporabnik ne vnese uporabniškega imena in gesla	Prijava zavrnjena	Prijava zavrnjena	7.9.2011
	Uporabnik vnese pravilno uporabniško ime in geslo	Prijava uspešna	Prijava uspešna	7.9.2011
Ogled stanja	Uporabnik na izbranem strežniku klikne zavihek in preveri stanje pomnilnika in diska	Enako stanje kot lokalno na strežniku	Stanje je enako	7.9.2011
Ponovni zagon sistema	Uporabnik na izbranem strežniku klikne gumb Restart in fizično preveri ali se je strežnik res ponovno zagnal	Strežnik se ponovno zažene	Strežnik se ponovno zažene	7.9.2011
Ugašanje sistema	Uporabnik na izbranem strežniku klikne gumb Shutdown in fizično preveri ali se je strežnik res ugasnil	Strežnik se ugasne	Strežnik se ugasne	7.9.2011
Zagon skripte	Uporabnik doda skripto, ki vrne trenutnega uporabnika tako, da izbere drugega uporabnika na seznamu	Rezultat je ime izbranega uporabnika	Vrnjen uporabnik je administrator, ki poganja program	7.9.2011
	Uporabnik doda skripto, ki vrne trenuten čas	Trenutni čas	Trenutni čas	7.9.2011

Testiranje pritiska

Testirano na strežniku s procesorjem AMD Athlon(tm) XP 1800+, 1Gb pomnilnika.

```
$ ab -n 1000 -c 100 http://localhost:4321/test
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking localhost (be patient)
Completed 100 requests
Completed 200 requests
Completed 300 requests
Completed 400 requests
Completed 500 requests
Completed 600 requests
Completed 700 requests
Completed 800 requests
Completed 900 requests
Completed 1000 requests
Finished 1000 requests


Server Software:        TornadoServer/2.0
Server Hostname:        localhost
Server Port:            4321

Document Path:          /test
Document Length:        4 bytes

Concurrency Level:      100
Time taken for tests:   5.439 seconds
Complete requests:      1000
Failed requests:        0
Write errors:           0
Total transferred:      109000 bytes
HTML transferred:       4000 bytes
Requests per second:    183.85 [#/sec] (mean)
Time per request:       543.931 [ms] (mean)
```

```
Time per request:       5.439 [ms] (mean, across all concurrent requests)
Transfer rate:          19.57 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    3   4.5      1      25
Processing:   154  532  67.9    551     631
Waiting:       87  489  81.6    512     563
Total:        154  534  69.5    552     646

Percentage of the requests served within a certain time (ms)
  50%    552
  66%    560
  75%    562
  80%    564
  90%    580
  95%    639
  98%    640
  99%    640
 100%    646 (longest request)
```