

**-** Specifikacija projektnog zadatka “Projektovanje i arhitektura sistema za rezervaciju sobe u hotelu Albatros” **-**

**Predmet: SE311 Projektovanje i arhitektura softvera**

Student:

Ivan Blagojević 2560

Beograd, 2017 godine

**Sadržaj**

[1. Uvod 3](#_Toc472995729)

[2. Arhitektura sistema 4](#_Toc472995730)

[3. Dizajn, arhitektura i projektovanje sistema 8](#_Toc472995731)

[3.1 Cilj razvoja ovog softvera 8](#_Toc472995732)

[3.2 Investitori projekta (stakeholders) 8](#_Toc472995733)

[3.3 Projektni proizvodi 8](#_Toc472995734)

[3.4 Metodologija projekta 9](#_Toc472995735)

[3.5 Interakcija sa naručiocima 9](#_Toc472995736)

[3.6 Filozofija testiranja 9](#_Toc472995737)

[3.7 Projektni tim/uloge 10](#_Toc472995738)

[3.8 UML dijagram 10](#_Toc472995739)

[3.9 Klasni dijagram 13](#_Toc472995745)

[3.10. Zachmanova matrica 14](#_Toc472995746)

[3.11. Baza podataka 16](#_Toc472995747)

[3.12. 4+1 Krutchenova pogleda 18](#_Toc472995749)

[4. Upotreba uzorka Visitor 23](#_Toc472995751)

[5. Upotreba uzorka Command 27](#_Toc472995752)

[6. Upotreba uzorka Iterator 32](#_Toc472995753)

[7. Mogućnosti i funkcije koje će biti testirane 35](#_Toc472995754)

[8. Mogućnosti i funkcije koje neće biti testirane 35](#_Toc472995755)

[9.Pristup/Strategija 36](#_Toc472995756)

[10. Kriterijumi za ulaz/izlaz iz faze testiranja](#_Toc472995757) 37

11.Sut prošao/Nije prošao kriterijum………………………………………………………………………………………………………….37

[12. Kriterijum obustavljanja testiranja i potrebni preduslovi za nastavak](#_Toc472995757) 38

[13. Potrebno osoblje i obuka](#_Toc472995757) 38

[14. Odgovornosti i održavanje](#_Toc472995757) 38

[15. Vremenski raspored 38](#_Toc472995757)

16. Test okruženje………………………………………………………………………………………………………………………………………39

[17. Budžet 40](#_Toc472995757)

[18.Rizici sistema 42](#_Toc472995758)

[19.Testiranje 43](#_Toc472995759)

[19.1 Plan testiranja 42](#_Toc472995760)

[19.2. Samo testiranje 42](#_Toc472995761)

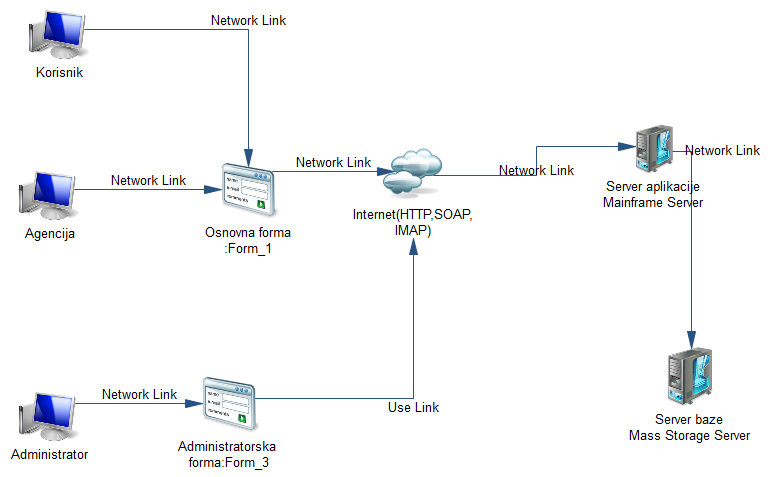
[20. Metrika 46](#_Toc472995762)

# 1. Uvod

Sama funkcija ovog sistem predstavlja osnovni sistem rezervacije hotela. Tačnije sama prva verzija sistema to zahteva, ostale verzije će poboljšavati sam sistem i truditi se da u skladu sa vremenom unapred sistem da bude što bolji i pristupačniji samim korisnicima. Sama rezervacija će biti omogućena svim korisnicima sistema koji imaju nalog, bez naloga sistem se neće moći koristiti. Pored same funkcije rezervacije, sistem će sadržati još neke funkcionalnosti. Tu će biti, naravno pored pravljenja naloga i prijavljivanja, pregled ponude hotela, izdavanja izveštaja za turističke agencije, provera slobodnosti soba za neki određeni datum i naravno sama funkcija otkazivanja rezervacije (koja će biti moguća samo 24h pre same rezervacije).

**1.1. Osnovni koncept**

Cilj ovog projektnog zadatka jeste sam razvoj aplikacije za rezervaciju hotela Albatros. Ovaj projekat će prikazati sve funkcije samog sistema, predstaviće pravljenje naloga (ovo je jedna od osnovnih funkcija i početnih, pošto korisnik neće moći da koristi i rezerviše sistem bez istog), predstaviće sam tok rezervacije, takođe predstaviće i otkazivanje i plaćanje rezervacije.Takođe sama aplikacija prikazaće i rezervacije sa strane turističkih agencija. Glavna razlika između običnih rezervacija i rezervacija od strane turističkih agencija jeste sam popust. Turistička agencija na svaki određeni broj dobija popust, koji je prethodno sam hotel odredio. Prednost ovog sistema jeste, sam način rezervacije (ovaj način dovodi do bržeg načina rezervacije i do smanjene papirologije prilikom iste), takođe I velika prednost je pregled dostupnosti hotela (tačnije prikaz da li u hotelu postoji sobodnih soba) i kao poslednja glavna prednost jeste lakši način plaćanja same rezervacije.



Slika 1. Osnovna schema rada online hotelske rezervacije

Sami korisnici putem računara šalju same zahteve za rezervaciju kroz određene kanale koristeći samu aplikaciju (web aplikacija) zajedno sa protokolima koji njoj pripadaju (HTTP, SOAP, IMAP). Sam server proverava da li su svi korisnički podaci dobri, pored toga sam server proverava da li je korisnik uspešno popunio rezervaciju i na kraju sama rezervacije se proslđuje.

# 2. Arhitektura sistema

Sama arhitektura se sastoji od hardverskih i softverskih komponenti koje zajedno čine samu rezervaciju u sistemu. Sama rezervacija predstavlja dvosmernu razmenu poruka. Ukoliko je rezervacija uspešna dobija se poruka da je soba u odabranom vremenu rezervisana, dok ukoliko je sam hotel popunjen i nema više slobodnih soba vraća se poruka da sama rezervacija za određeni datum nije moguća.

**2.1. MVC (model-view-controller) arhitektura**

MVC je skraćenica za Model – View – Controller arhitekturu. Radi lakšeg razvoja samog projekta i primene projekta, sama veb – aplikacija je podeljena u 3 osnovna modela:

* Model – odgovoran je za rukovanje u interakciji baze podataka i kontrolera
* Pregledač – odgovoran je da prikazuje sadžaj korisnicima
* Kontroler – odgovoran za postupanje po akciji korisnika

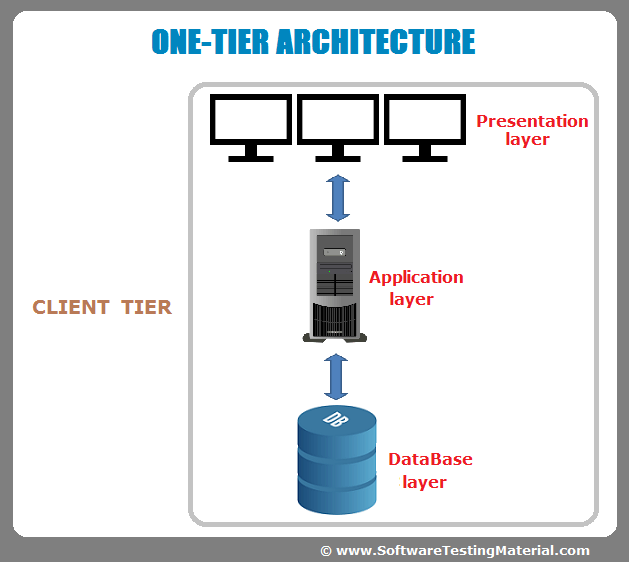
Ukoliko želi korisnik da pogleda dostupnost sobe za određeni dan to radi na sledeći način:

* Model – preuzima odabran datum iz baze podataka
* Pregledač – Prikazuje box sa svim slobodnim slobana
* Kontroler – Šalje poruku sobe koju je korisnik odabrao

Sama arhitektura je uglavom regulisana putem kontrolera. Krajni korisnik na osnovu kontrolera zahteva model. Model preuzima podatke iz baze i na kraju na osnovu odgovora dobijenih promenom kontrolera, modifikuju se podaci u pregledaču

**2.1.1. MVC (model-view-controller) jednoslojna arhitektura**

Klijent – server arhitektura u kojoj se svaka komunikacija elemenata (prezentacioni sloj, poslovna logika i sloj baze podataka) se nalaze u jednoj jedinici.



Slika 2.1.1.1. Predstavlja osnovni izgled jednoslojne arhitekture

Svaka jedinica odgovorna za klient – server interakciju se nalazi (postavljena je) na jednom mestu i to u klijentu. Sama arhitekturu je moguće ponoviti za server isto kao i za n – broj klihenata. Ovaj tip aražmana se smatra najjednostavnijim i najdirektnijim. Lak pristup bazi podataka isto kao i poslovnoj logici se povećava.

Prednosti:

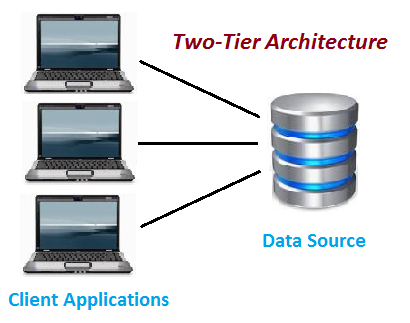
* Bolje performance: pošto je ovakva arhitektura, tačnije ugrađena je, ona smanjuje mogućnost gubitke pristupa ili samog gubitka konekcije
* Veća sigurnost: sama baza je takođe ugrađena u ovoj arhitekturi, to jeste klijent koji želi da pristupi podacima i poslovnoj logici ima sva prava.

Nedostaci:

* Kako su svi slojevi instalirani kod svakog klijenta, troškovi za sam razvoj su povećani.
* Troškovi održavanja su takođe povećani.

**2.1.2. MVC (model-view-controller) dvoslojna arhitektura**

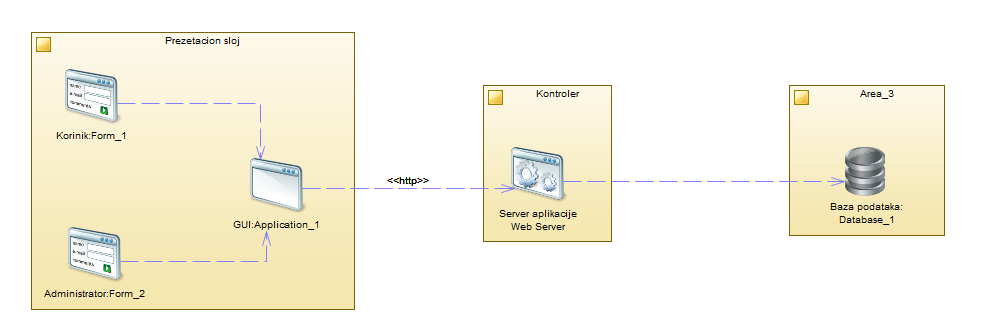
I ovoj arhitekturi, GUI (pogledni modul) se nalazi na strani klijenta, dok se sama poslovna logika (kontroler) i baza podataka (modul) nalaze na strain server. Klijent šalje zahtev za server preko pregled modula i aplikacija na strain server direktno komunicira sa bazom podataka i preuzima rezultate koji se šalju nazad do klijenta. Komunikacija između klijent –servera zagteva stalnu internet konekciju.



Slika 2.1.2.1. Predstavlja osnovni izgled dvoslojne arhitekture

U našoj aplikaciji, aplikaciji za rezervaciju sobe u hotelu Albatros, koristićemo ovu arhitekturu. Uzećemo u obzir da korisnik (pojedinačni korisnik ili agencija) želi da rezerviše sobu broj 10 od 20.1. do 22.1. Ovaj zahtev će biti u red i biće na čekanju na server sve dok se informacije ne budu proverene u samoj bazi.

Na sledećoj slici prikazaćeno našu arihitekturu sistema.



Slika 2.1.2.2. Dijagram arhitekture aplikacije za rezervaciju karata

Kao što možemo videti na samoj slici 2.1.2.2, na kojoj je prikazana arhitektura naše aplikacije, prikazan je dijagram dvoslojne arhitekture. Kao što možemo videti sistem će činiti sama klijentska i serverska strana. Na klijentskoj strani imamo korisničku formu i agencijsku formu kao i sam gui koji predstavljaju sam prezentacioni sloj (View). Nakon samih unetih podataka, koji su potrebno za rezervaciju hotelske sobe, prelazi se na sam server. Pre samog komuniciranja sa bazom, sistem putem kontrolera prihvata sve podatke i šalje ih u bazu. Nakon toga, sama baza podataka odgovara na sve serverske upite.

Prednosti:

* Brža komunikacija: Pošto je GUI na strani klijenta, održavanje i modifikacija same baze je laka
* Manji troškovi za sam razvoj

Nedostaci:

* Konstantna internet konekcija je potrebna.
* Kako je aplikacija namenjena za veći broj korisnika, moguće je doći do zastoja u samoj obradi zahteva klijenta.

**2.1.3. MVC (model-view-controller) troslojna**

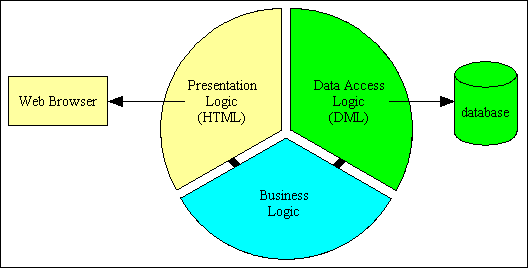
**arhitektura**

U troslojnoj arhitekturi imamo dva server i klijent koji su uključeni. MVC je podeljen na sledeći način:

* Model – Baza servera
* Pregledač – Klijentska mašina
* Kontroler – Server aplikacije

Klijent komunicira sa serverom aplikacije u obliku zahteva. Ovi zahtevi služe serveru u vidu odgovara. Odgovori uključuju pronalaženje podataka iz baze podataka na serveru aplikacije.

Sama arhitektura izgleda ovako:



Prednosti:

* Više sigurnosti
* Ako jedan od tri dela prestane raditi ostalu moduli mogu i dalje funkcionisati

Nedostaci:

* Povećana složenost
* Razdvajanje dovodi do problema sa performansama

2. Glavne prednosti MVC modela u odnosu na ostale arhitekture su:

* Brži process razvoja – MVC podržava brz i paralelan razvoj. Korišćenjem MVC modela, prilikom razvoja veb aplikacije, dovodi do mogućnosti da jedan programmer radi na prikazu dok drugi može raditi na kontroleru kako bi stvorili poslovnu logiku same veb aplikacije. Samim time razvoj aplikacije korišćenjem ovog modela može dovesti da se aplikacija završi tri puta brže od aplikacije koja je razvijena nekim od drugih razvojnih obrazaca.
* Mogućnost da se obezbedi više prikaza – U samoj ovoj arhitekturi, možemo napraviti više prikaza za model. Danas, postoji velika potražnja za nove načine da se pristupi aplikaciji i zbog toga MVC razvoj je odlično rešenje. Osim toga, u ovoj metodi, dupliranje koda je vrlo ograničeno, zato što se odvajaju podaci i poslovna logika sa ekrana.
* Podrška za asihronu tehniku – MVC arhitektura može da se integriše sa JavaScript Framework-om. To znači da sama MVC aplikacija može da radi i sa PDF fajlovima i desktop widgetima. MVC takođe podržava asihrone tehnike, koje pomažu programerima da razviju aplikaciju koja se učitava brzo.
* Izmene ne utiču na ceo model – Za bilo koju veb aplikaciju, sam korisnički interfejs ima tendenciju da se često menja čak i više od biznis modela kompanije. Očigledno je da se često prave promene boja, fonta, rasporeda na ekranu pa i samo dodvanje novih platformi u samoj aplikaciji. Osim toga, samo dodavanje pogleda je veoma lako u MVC paternu, zato što Model deo ne zavisi od Pogled dela. Stoga, bilo kakve promene u modelu neće uticati na celu arhitekturu.
* MVC model vraća podatke bez formatiranja – MVC vraća podatke bez ikakvih promena. Dakle, iste komponente moogu da se koriste i pozivaju za korist bilo kog interfejsa. Na primer, bilo koja vrsta podataka može se formatirati sa HTML, ali se ist može formatirati I sa Macromedia Flash ili Dream viewer.

# 3. Dizajn, arhitektura i projektovanje sistema

Sam softver čiju smo arhitekturu prikazali u prethodnom delu dokumnta jeste softver za rezervaciju sobe u hotelu Albatros. Korisnici sistema su ili individualni korisnici ili turističke agencije. Pored samih korisnika sistema, sistem je namenjen i zaposlenima hotela, koji će imati određene privilegije u zavisnosti od funkcije. Naravno kada su u pitanju sami stariji korisnici sistema, koji su navikli na tradicionalni način rezervacije (rezervacije putem telefona), ljudi na recepciji će moći putem određene forme da naprave rezervaciju za datum i sobu koju je korisnik odabrao. Glavna funkcija samog sistema jeste rezervacija, korisnici će moći da kada se prijave na sistem, uneti potrebene informacije (informacije o sobi, datumu, vrste sobe, mogućnost TV, mogućnost pogleda na more, mogućnost pušenja u sobu) i na kraju putem sms poruke povrditi samu rezervaciju. Druga isto jako bitna funkcija jeste i samo plaćanje putem kreditne kartice (korisnik takođe unosi sve potrebne informacije). Sami administrator će jedini imati mogućnost da dodaju nove zapošljene u bazu i jedini će moći da dodaju sobu ukoliko u budućnosti dođe do proširenja hotela.

Ovaj sistem pomaže korisnicima da na lak i brz način rezervišu sobu i da na lakši način sama uprava hotela može videti sve rezervacije i na taj način smanili nepotrebnu papirologiju.

## 3.1 Cilj razvoja ovog softvera

Cilj samog softvera jeste smanjena papirologija putem rezervacije, naravno pored glavnog cilja, koji je izvšavanje rezervacije putem interneta. Sistem će koristiti ne samo korisnici hotela, već i samim zaposlenima u hotelu. Nakon samog završetka sistema, korisnici su ustanovili da je sam softver jako koristan. Na ovaj način su sami korisnici krenuli putem online softvera, nakon izbacivanja softvera.

## 3.2 Investitori projekta (stakeholders)

Sami stakeholder sistema jeste hotel Albatros koji je postavio sve zahteve i na kraju sa svim osobljem koji su bili zaduženi za razvoj došli do dobrog softvera.

## 3.3 Projektni proizvodi

*Project Plan* – predstavlja dokumentaciju samog projekta. Ova sama dokumentacija sadrži detalje zadatka projekta, sve potrebne resurse i sve planove realizacije.

*Business Blueprint –* predstavlja dokumentaciju konačne verzije korisničkih zahteva i sve modele poslovnog procesa.

*Hardver* – specifikacija hardvera koji je potreban da bi sam sistem funkcionisao. (ovaj sistem ne zahteva neku specifičku hardversku strukturu)

*Korisnička dokumentacija* – svaki softver mora imati korisničko uputstvo, ono služi korisnicima kao pomoć za upotrebu sistema.

## 3.4 Metodologija projekta

Na samom početku, pre početka samog rada na projektu, moramo odrediti timove i samog vođu koji će voditi računa i nadgledati sve faze razvoja. Nakon ove faze, naravno u ovoj fazi spadaju sve funkcije i same implementacije istih, vrši se testiranje koje jako bitno. Posle svih prikupljenih podataka i svim rezultatima vođa predaje sam plan o daljem razvoju softvera. Nakon faze broj jedan, prelazimo na sledeću istu jako bitnu fazu, a to je obučavanje korisnika za korišćenje sistema. Sama sledeća faza označava kako korisnik da na svoju ruku pronađe rešenje za problem (funkciju) putem korisničkog uputstva.

## 3.5 Interakcija sa naručiocima

Sama interakcija sa naručiocima se vrši u nekoliko faza. Sama početna faza obuhvata interakciju za definisanje zahteva korisnika softvera, naravno u ovaj fazi se postavljaju funkcionalni u nefunkcionalni zahtevi, cena softvera I sve teme za početak rada na softveru. Sama sledeća faza jeste poboljšanje sistema. Ukoliko se u sistemu javila neka greška, koju nakon testiranja nije moglo uočiti, sami dizajneri sistema postavljaju datum i puštanje nove verzije sistema. Sama sledeća faza jeste samo dogovor sa naručiocima kako sistem poboljšati i kako napraviti bolju verziju postojećeg sistema.

## 3.6 Filozofija testiranja

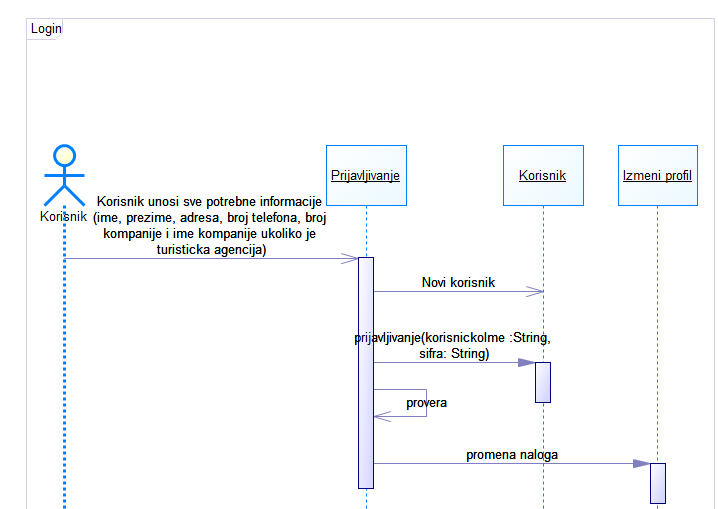
Samo testiranje će obavljati testeri zajedno sa menadžerom, koji će biti tu da nadgela i prati sve testove nad sistemom. Menadžer takođe vodi i brigu o samom odražavanju i kvalitetu softvera .Samo testiranje se sastoji od više faza i zavisi od koliko modula se nalazi u softveru. (tabela 2 predstavlja moguće greške do kojih je moguće doći.) Kao prva faza u svakom testiranju, jeste postavljanje testova koje je potrebno izvršiti. Nakon što smo postavili sve test slučajeve i sve očekivane rezultate, krećemo samo testiranje. Naravno, svaki test slučaj dokumentujemo i vršimo analizu kako rešiti taj problem na koji smo naišli. Najbolji rezultat testiranja ostvarujemo putem zajedničkog testiranja (u prevodu znači da programeri vrši prilikom implementacije testiranje i na taj način olakšavaju testerima posao) .

## 3.7 Projektni tim/uloge

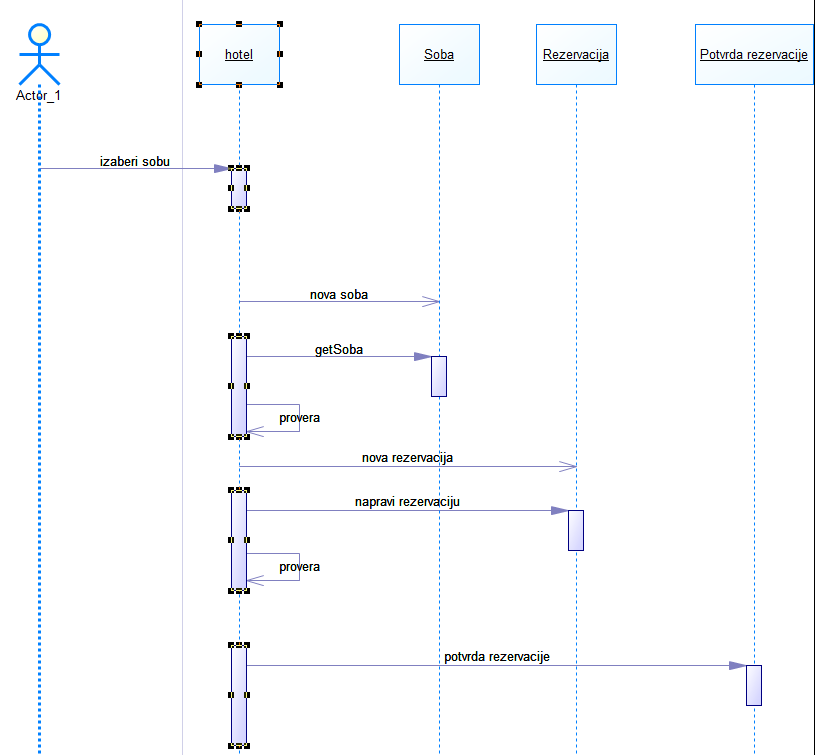
3. U sledećoj delu prikazaćemo organizaciju projekta i uloge koje će biti dodteljenje. Same ove ulogu mogu da se dopune, uklone i promene na kraju može doći do promene odgovornosti.

* Menadžer projekta – pravi plan, upravlja isporučenju prema planu, regrut projektnog osoblja, vode i upravljaju timom. Takođe određuju metodologiju koja je korišćena na projektu, postavlja raspored projekta i određuju svaku fazu.
* QA menadžer – vodiće detaljnu dokumentaciju o proizvodu, koje su performance, koja je namena i koje su pronađene greške i propusti
* Biznis analitičar – pomaže u definisanje projekta, prikuplja zahteve, uverava se da rezultati projekta ispunjavaju uslove. Takođe daju test rešenja za proveru ciljeva.
* Razvojni tim – razume posao koji treba da se završi, planiraju aktivnosti sa više detalja, obavljaju posao u skladu sa budžetom, vremenskim okvirom i očekivanim kvalitetom. Takođeinformišu menadžera ukoliko dođe do problema ili rizika.
* Test menadžer – nadgleda rad testera, daje predloge mogućih testova i beleži sve pronađene greške koje su pronađene
* Tester – odgovorni za testiranje komponenti korisničkih interfejsa.

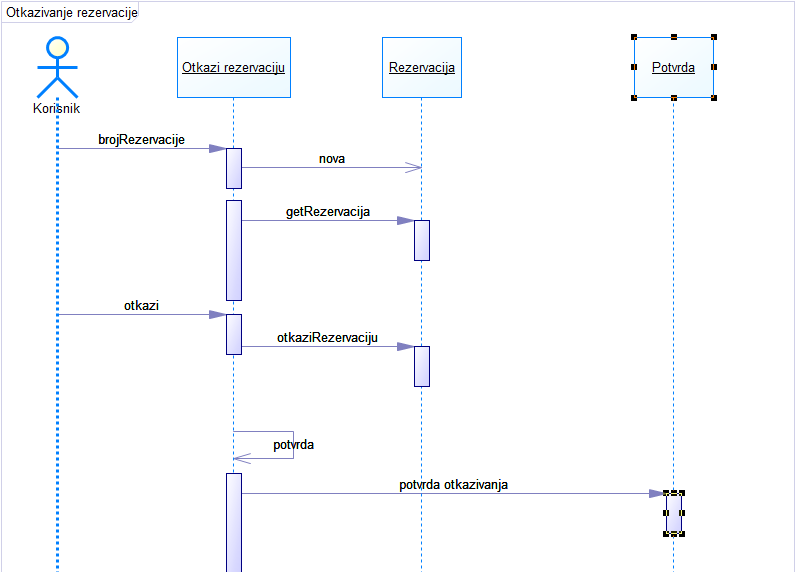
## 3.8 UML dijagram

U sledećem izlaganju daćemo opis UML dijagrama koji predstavlja sliku suštinske funkcionalnosti sistema za rezervaciju sobe u hotelu. Predstavićemo sekvencijalni diagram samog sistema i tako korisniku prikazali samo funkcionisanje sistema.

**Slika 3.8.1. Predstavlja sekvencijalni dijagram za login**

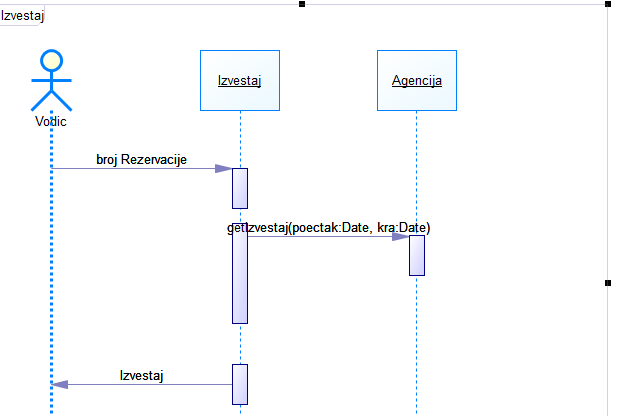
Korisnik pristupa sistemu i unosi sve potrebne informacije koje mu forma traži i na taj način pravi nalog i ostvaruje mogućnost korišćenja sistema. Nakon što je sistem napravio nalog, daje korisniku mogućnost da pristupu sistemu unošeći korisničko ime i šifru. Ukoliko su podaci validni korisnik pristupa sistemu i dobija mogućnost promene svog naloga. Ukoliko je pogrešio, proces se ponavlja dok ne unese validno korisničko ime i šifru. 

## Slika 3.8.2. Predstavlja sekvencijalni dijagram za rezervaciju sobe u hotelu

Korisnik sistema bira sobu, sistem šalje poruku sa zahtevom za novu sobu, nakon toga korisnik bira sobu i sistem proverava samu slobodnost te sobe. Ukoliko je slobodna soba, sistem pokreće proces rezervacije, korisnik pravi rezervaciju. Sistem nakon naravljene rezervacije, proverava da li su sve potrebne informacije tu. Ukoliko jesu korisnik dobija mogućnost potvrde rezervacije.

## Slika 3.8.3. Predstavlja sekvencijalni dijagram za otkazivanje rezervacije

Korisnik prilikom otkazivanja rezervacije mora imati sam broj rezervacije. Korisnik unosi sam taj broj rezervacije i sama funkcija se šalje sistemu. Dobijaju se sve informacije o sistemu, ukoliko je korisnik siguran bira opciju otkaži kako bi otkazao rezervaciju. Sistem potvrđuje da li moguće otkazati (rezervaciju nije moguće otkazati 24h pre checkIn dana). Ukoliko je ispunio osnovni uslov, korisnik otkazuje rezervaciju.

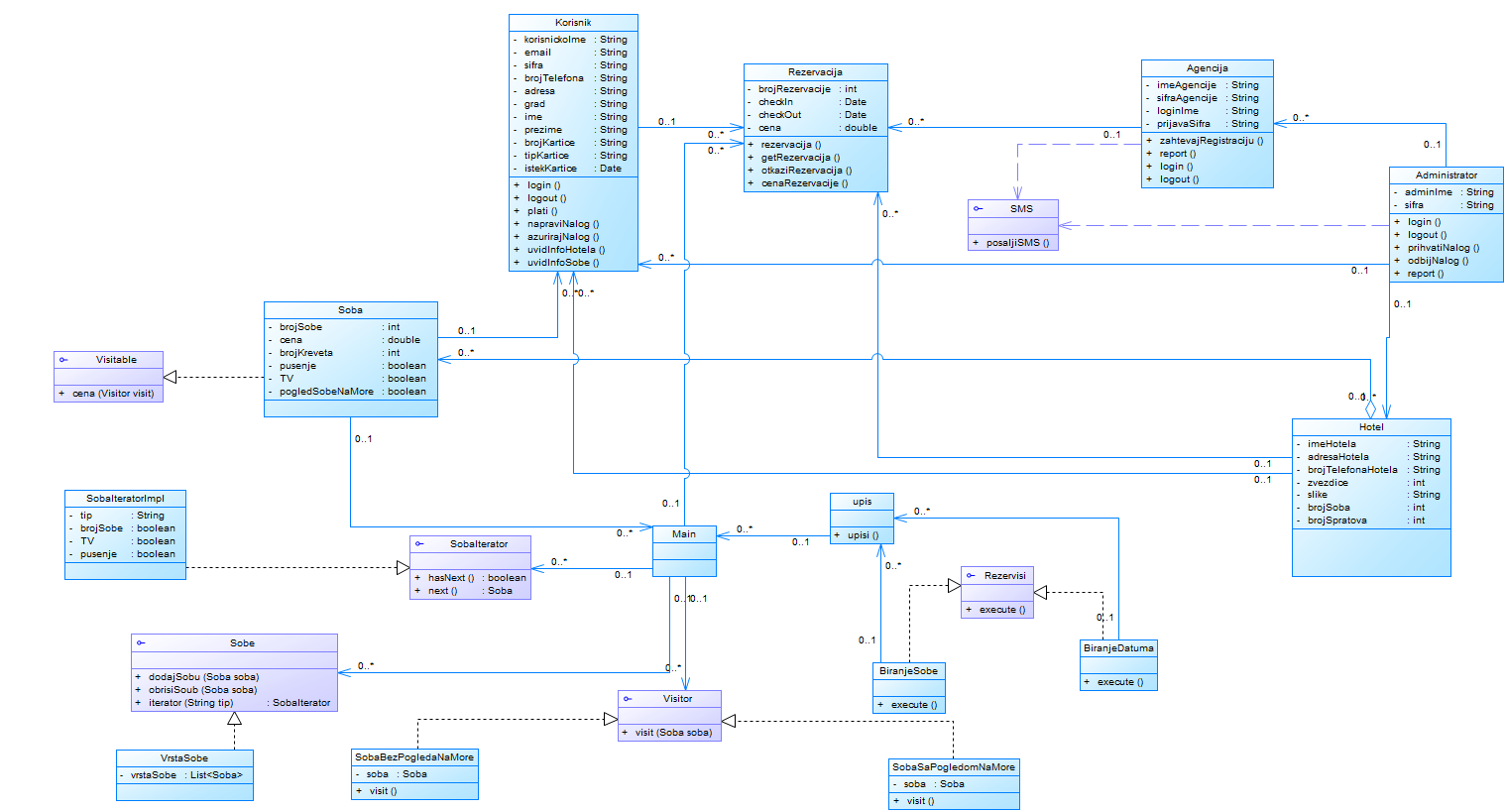


**Slika 3.8.4. Predstavlja sekvencijalni dijagram za izveštaj za turističku agenciju**

Ovde vidimo da vodič pomoću broja rezervacije, sistem prikuplja podatke o informacijama rezervacije, dobija celokupni izveštaj turističke ture.

## 3.9 Klasni dijagram

U ovom delu biće prikazan klasni dijagram, koji je dizajniran putem PD. Na slici 3.9.1. imamo prikazan klasni dijagram sistema. Sam dijagram se sastoji od nekoliko klasa. Prva klasa jeste klasa korisnik. Sama klasa se sastoji od nekoliko atributa i funkcija. Svaki korisnik sistema će imati svoje korisničko ime, šifru, brojTelefona, adresu, grad, ime, prezime brojKartice, tipKartice i datum isteka kartice. Pored osnovnih atributa tu su i funkcije, login, logout, napraviNalog, azurirajNalog, uvidUInfoHotela i uvidInfoSobe. Sama klasa je povezana asocijacom sa klasama Soba i klasom Hotel. Razlog toga jesu same funkcije koje korisnik ima, a to su uvidInfoHotela i uvidInfoSobe. Korisnik sa ovim funkcijama može da koristi sistem kako treba, tačnije može da vidi sve informacije o hotelu i da vidi sve informacije o sobi. Samom asocijacijom u klasi korisnik pravimo instancu klase Soba i Hotel i na taj način putem get metode uzimo informacije potrebne za naše ove dve funkcije. Pored toga korisnik je takođe povezan sa klasom rezervacija. Takođe i ova klasa je povezana asocijacijacijom sa klasom rezervacija. Samo što u ovom slučaju pravimo instancu klase Osoba i tako prilikom pravljenja rezervacija uzimamo uz pomoć get metode informacije potrebne za funkciju rezervacija. Kod klase rezervacija koristimo command. Njega ćemo kasnije detaljno objasniti ali glavna funkcija jeste samo upisivanje rezervacije. Sama klasa rezervacija je takođe povezana asocijacom sa još dve klase. I to su klase Hotel i Agencija. Ovim putem pozivamo instance hotel i agencija i tako prilikom rezervacije uz pomoć get metode postavljamo u kom hotelu se radi sama rezervacija, koja se soba rezerviše (uzima se pomoču get metode broj sobe) i takođe isto kao i korisnik tako se i postavlja koja je agencija naravila rezervaciju.Sledeća klasa koja nije objašnjena jeste klasa Agencija, ona je pored već opisanih veza, povezana asocijacijom sa klasom Admnistator. Na taj način poziva se instanca Agencija koja u klasi administrator pomaže za funkciju prihvati nalog i report. Kada agencija zahteva nalog, administrator mora da potvrdi. Osim toga agencija kada zahteva report, putem SMS-a se vrši autentifikacija, sama klasa SMS je klasa sama za sebe i povezana je sa dependacy sa klasom agencija i administrator. Ukoliko se slaže sam autentifikacioni kod, administrator šalje report agenciji. Naravno tu je i klasa Administrator, njene veze smo već objasnili, osim veze sa klasom Hotel i klasom Korisnik. Asocijacom su povezane iz razloga što ukoliko želimo da razvijemo lanac hotela, putem ovoga možemo, pozivanjem instance klase Hotel, da postavimo koji je administrator zadužen za koji Hotel, takođe sa klasom Korisnik je povezan pošto administrator prilikom prihvatanja naloga korisnika uzima njegove atribute i na taj način odobrava ili odbija nalog. I na kraju objasnićemo još jednu vezu klase Hotel, ostale veze su prethodno objašnjene, a to je veza sa klasom Soba. Sama veza predstavlja agregaciju, to je najlakše objasniti Hotel ima Sobe. Kao što vidimo slici ovo je klasni dijagram, nakon implementiranja novih verzija doći će do poboljšanja sistema i sam klasni će doživeti određene promene.



Slika 3.9.1. Klasni dijagram sistema

Na samom klasnom dijagramu možemo primetiti da imamo sledeće paterne:

* Command
* Iterator
* Visitor

Jedan od prvih paterna koji smo primenili u ovom projektu jeste Visitor patern. Sam patern zahteva sam interfejs Visitor koji će definisati operacije posetilaca klase. Pored ovog interfejsa imamo i sam interfejs Visitable koji definiši funkciju cena. Sama klase SobaSaPogledomNaMore i SobaBezPogledaNaMore implementiraju sam interfejs Visitor, dok sama klasa Soba implementira interfejs Visitable. Detaljnije opisan kod, kao i prikaz prednosti i mana ovog paterna biće predstavljeno u poglavlju broj 4.

Sam command pattern koristimo za upis rezervacije. Kao što vidimo na dijagramu napravili smo interfejs rezerviši koji se ponaša kao komanda. Pored samog interfejsa koji je stvoren u ovom paternu, imamo još dve dodate konkretne komandne klase a to su BiranjeSobe i BiranjeDatuma koje implementiraju sam interfejs Rezerviši. Ove dve klase obrađuju same komande. Takođe u samom ovom patrenu dodali smo i klasu upis, ova klasa će nam predstavljati a i ponašaće se kao sam zahtev. Samo dodatno opisivanje i detaljnu implementaciju ovog paterna objasnićemo u poglavlju broj 5.

Pored samog visitor i commaand paterna koje smo već naveli tu je i iterator patern. Na dijagramu možemo uočiti da imamo interfejs SobaIterator koja navodi navigaciju metoda. Pored samog tog interfjesa imamo i interfejs koji pored svojih funkcija sadrži i implementira sam iterator. Takođe imamo i samu klasu vrstaSobe koja implementira samu klasu soba. U ovoj klasi, implementiramo i postavljamo same funkcije koje se nalaze u sam interfejs soba. Samu implementaciju paterna, predstavićemo u poglavlju broj 6.

## 3.10. Zachmanova matrica



Slika 3.10.1. Predstavlja Zahmanov Framework

Zahmanov Framework predstavlja osnovnu strukturu za arhitekturu preduzeća, koja pruža formalin struktuirani način gledanja i definisanja preduzeća. Ontologija je dvodimenzionalna klasifikacija šema koja održava raskrsnicu između dve istorijske klasifikacije. Prvi su primitivni upiti:

* Šta
* Kako
* Kada
* Ko
* Gde
* Zašto

Dok su druge izvedene iz filozofskog koncepta refikacije, transformacije apstraktne ideje u jednu instancu[1]. Te refikacije:

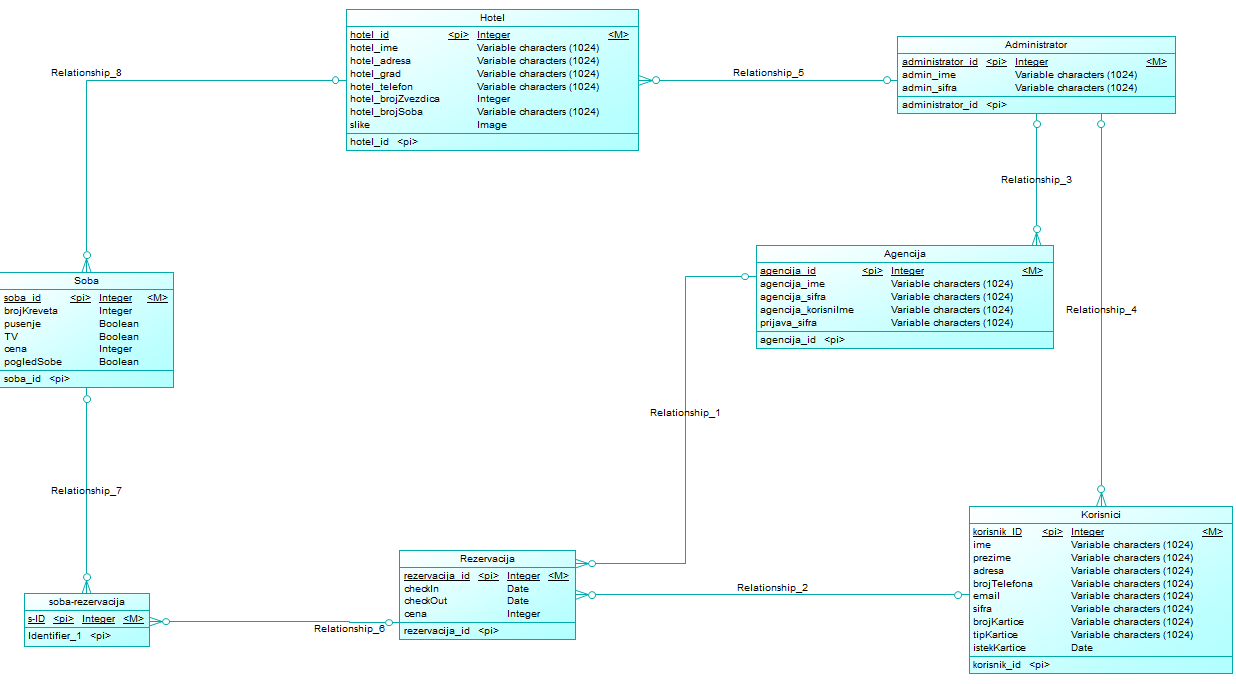
* Indentifikacija
* Definicija
* Predstavljanje
* Specifikacija
* Konfiguracija
* Primeri

Sam Framework ima svoja pravila i to su[2]:

* Pravilo 1: Kolone su zamenljive ali se ne mogu smanjiti ili dodati
* Pravilo 2: Svaka kolona može imati svoj meta-model
* Pravilo 3: Svaki model od svake kolone, veza objekta i struktura su jedinstveni
* Pravilo 4: Svaki red opisuje pogled na određene poslovne grupe i jedinstven je za njega
* Pravilo 5: Kombinacija 2,3 i 4 mora da proizvede jedinstvenu ćeliju gde će svaka predstavljati poseban slučaj.
* Pravilo 6: Menjanje imena može proeniti osnovnu logiku okvira i zato nije poželno menjati ga
* Pravilo 7: Logika je rekurzivna. Logika je relacija između dva primera istog entiteta.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | ZAŠTO | KAKO | ŠTA | KO | GDE | KADA |
| KONTEKSTUALNI | Postavljanje zahteva | Implementacija, dokumentacija, dizajn, testiranje | Informacije o uslugama hotela i samom hotelu | Upravnici hotela | Prostorije hotela | Prve faze projkte, prve 2-3 nedelje |
| KONCEPTUALNI | Odnosi se na bazu, sve njene relacije i entite možemo videti u sledećem poglavlju. | Predstaljava biznis proces model | Samu bazu možete videti u sledećem delu. | Programeri i menadžeri | Nalazi se na samom serveru baze. | Predstavllja biznis proces model- |
| LOGIČKI | Sami uslovi će biti naknadno dodati. | Same moguće arhitekture su već prikazane u prethodnom delu. | Data flow dijagra predstavljen u prethodnom delu | Isto kao i uslovi, biće naknadno dodati. | Arhitektura je takođe prikazan u prethodnom delu. | Use case delovi su prikazani u prethodnom delu. |
| FIZIČKI | Sami uslovi će biti naknadno dodati. | Prikazano u prethodnom poglavlju. | Samu bazu možete videti u sledećem delu. | Sam prototip aplikacije u fazi obrade. | Arhitektura je prikazana u prethodnom delu | U fazi testiranja. |
| VANKONTEKSTUALNI | Sama ciljani uslovi će naknadno biti dodati. | Opisano i postavljeno na početku rada projekta | Samu bazu možete videti u sledećem delu. | Sam interfejs će naknadno biti dodat | Arhitektura je prikazana u prethodnom delu | Use case delovi su prikazani u prethodnom delu. |

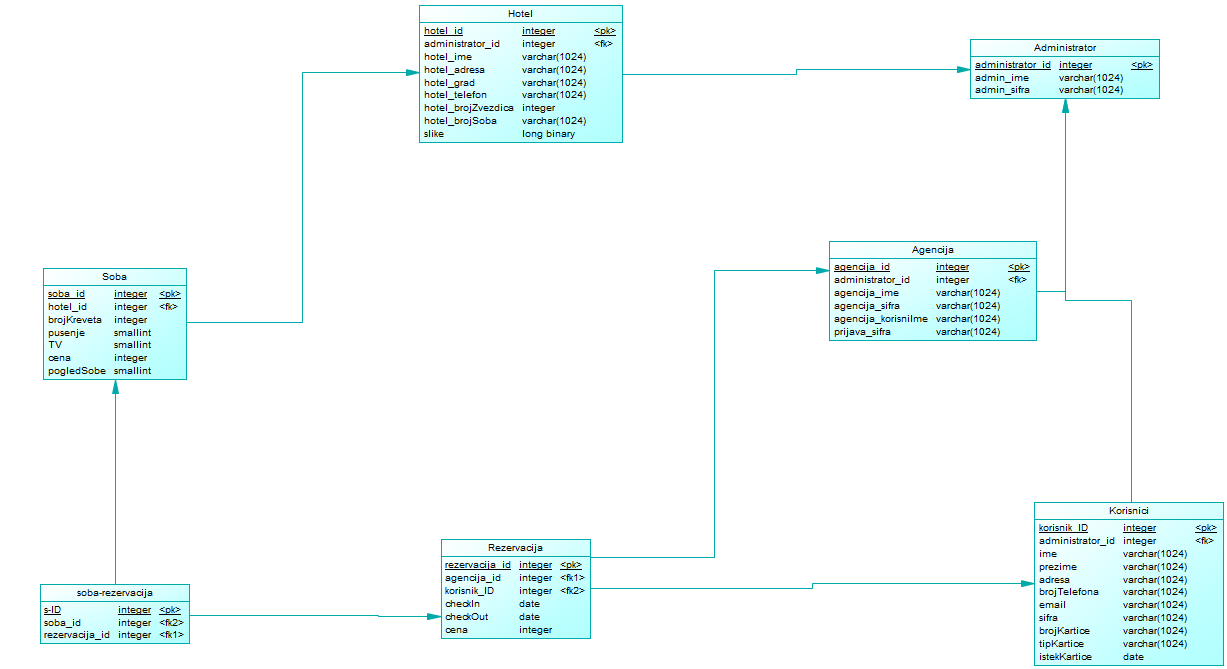
## 3.11. Baza podataka



Slika 3.11.1. Konceptualni model baze podataka

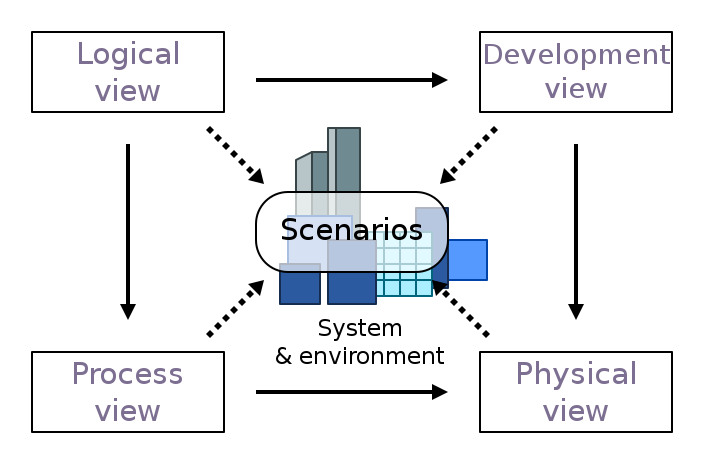
## Na slici možemo videti bazu podataka. Ona se sastoji od sledećih entiteta:

* Hotel – sve informacije Hotela se nalaze ovde, isto kao i administratorskiID koji je zadužen za hotel
* Sobe – entitet koji će u sebi imati brojSobe kao ID, ukupan broj soba i strani kljuc soba (sve specifikacije sobe će se naći u ovom entitetu)
* Soba – entitet koji sadrži sve specifikacije o mogućim sobama u hotelu
* Rezervacija – entitet koji će imati od- do datum zajedno sa id korisnika ili agencije u zavisnosti od klijenta koji rezerviše
* Korisnici – entite sa svim informacijama o korisnicima, kao i podacima za pristup sistemu
* Agencija – entitet sa registarskim brojem agencijeske firme
* Soba rezervacija – entitet sa svim podacima o rezervaciji. Koju sobu u tom hotelu i u kom vremenskom periodu je korisnik rezervisao
* Administrator – entitet sa potrebnim podacima za pristup sistemu.



Slika 3.11.2. Fizički model baze

## 3.12. 4+1 Krutchenova pogleda

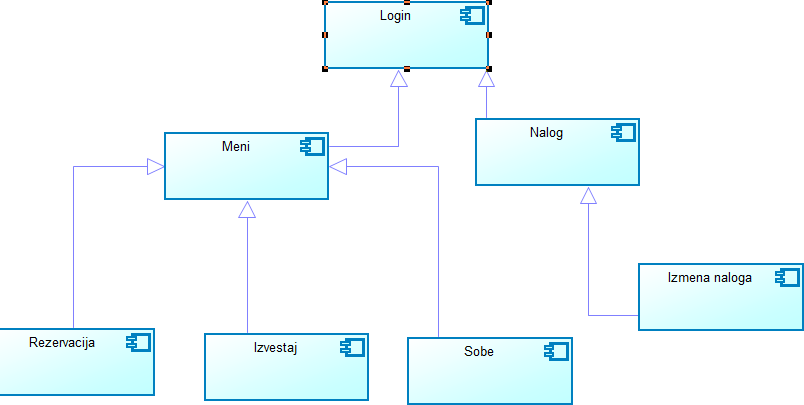


## Slika 3.12.1. Predstavlja 4+1 Krutchenova pogled

Kao što možemo videti na slici, slika predstavlja model koji je dizajnirao Philippe Krutchen, 4+1 pogled se sastoji od pet pogleda. Prvi model predstavlja razvojni, drugi fizički, treći logični a četvrti procesni pogled. Na kraju sam +1 predstavlja scenarije korišćenje. Sam 4+1 pogled primenićemo u našoj aplikaciji. U razvojnom pogledu predstavićemo sam dijagram komponenti koji je vezan za našu aplikaciju. U fizičkom pogledu predstavićemo sam razvojni dijagram koji se isto tiče naše aplikacije. Nakon drugog pogleda, prikazaćemo logički pogled koji će sadržati dijagrame aktivnosti i stanja naše aolikacije i na kraju ćemo predstaviti procesni pogled koji će biti sačinjen od dijagrama aktivnosti. Sam +1 deo, tačnije scenarije, prikazaćemo na kraju u vidu use case dijagrama.

**Development View**

Ovaj razvoj predstavlja sistem iz perspektive samog programera, bavi se samo upravljanjem softvera, takođe ovaj razvoj je poznat kao sam implementacioni pogled. UML diagram koji se koristi jeste Compoonent diagram pomoću kojeg ćemo predstaviti i opisati same komponente sistema. [3]

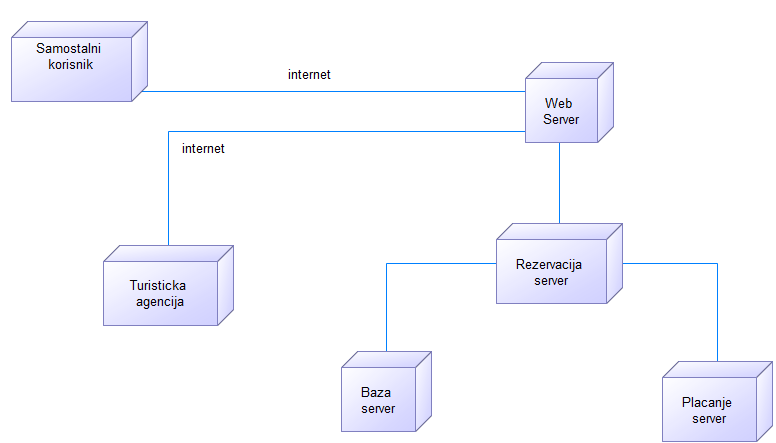


Slika 3.12.2 – Dijagram komponenti

Na samom dijagramu možemo videti koliko će naša prva verzija sistema imati komponenti. Vidimo da ćemo imati samu login komponentu, naravno bez nje dalje korišćenje sistema i nije moguće, nakon koje će moći da se bira neka od sledećih opcija (u programesku svetu, izabere sledeća komponenta).

**Psyhical View**

U ovom razvoju prikazujemo sistem kako bi ga video softverski inženjer. Sam dijagram koji se ovde koristi jeste deployment dijagram[4].

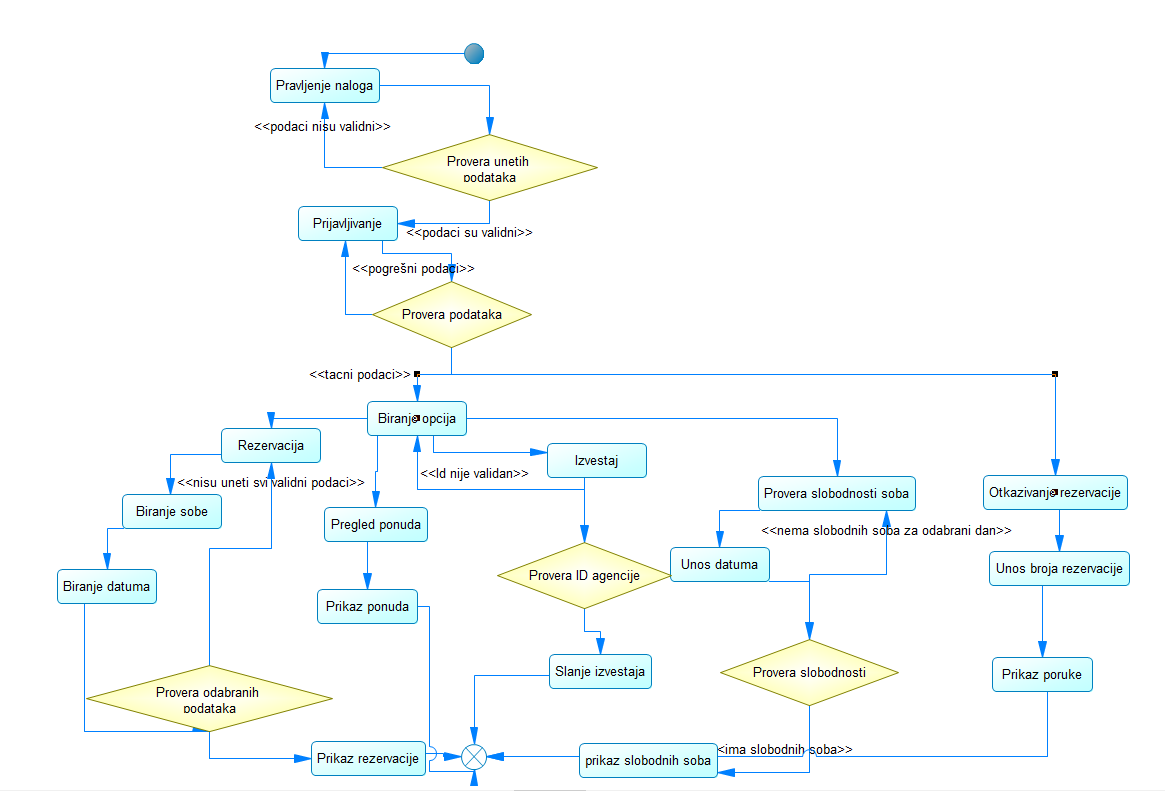


Slika 3.12.3 – Razvojni dijagram

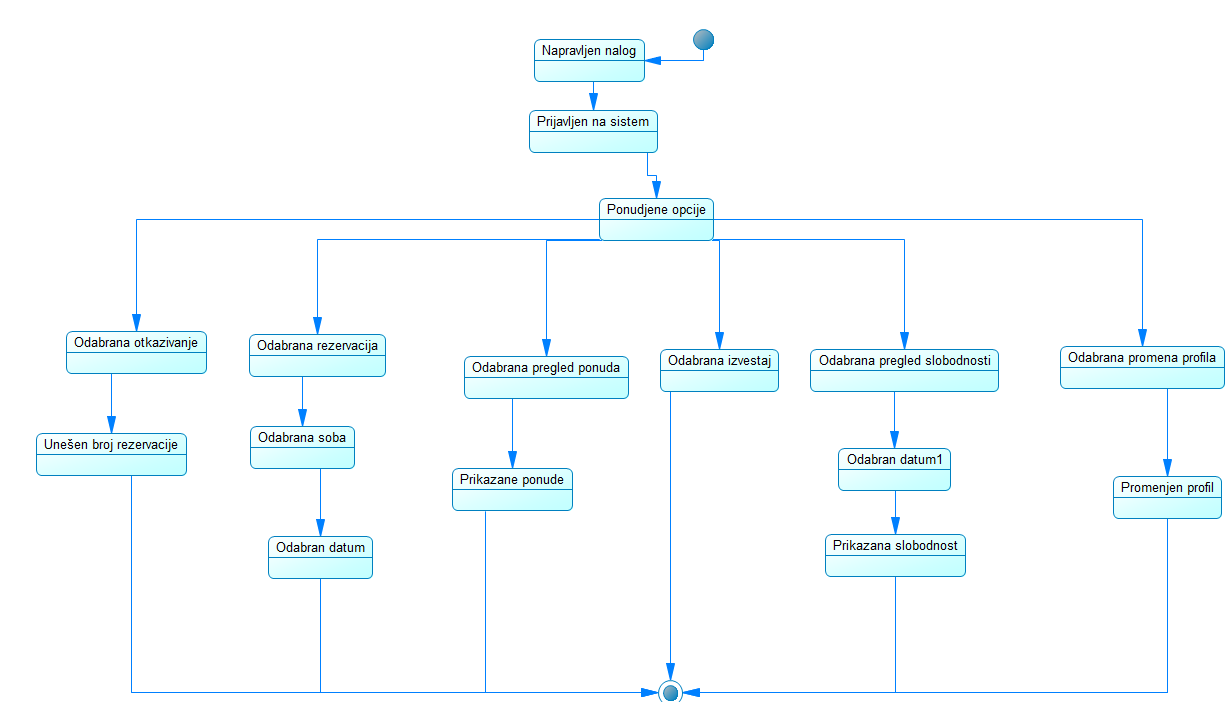
Na samom dijagramu možemo videti da korisnici (samostalni i turistička agencija) putem internet komuniciraju sa web serverom. Takođe vidimo da sam web server komunicira sa serverom za rezervaciju, sa serverom koji čuva bazu podataka i sa serverom za plaćanje (u fazi je razvoja, u nekoj od narednih verzija će biti uljučen u sam sistem).

**Logic View**

Ovaj razvoj se bavi funkcionalnošću koji će korisnici imati. Korišćeni UML dijaagrami su dijagram aktivnosti, klasni dijagram i dijagram stanja. Pošto smo u prethodnom delu predstavili klasni dijagram u ovom delu predstavićemo preostala dva dijagrama[4].



Slika 3.12.4. Predstavlja dijagram aktivnosti



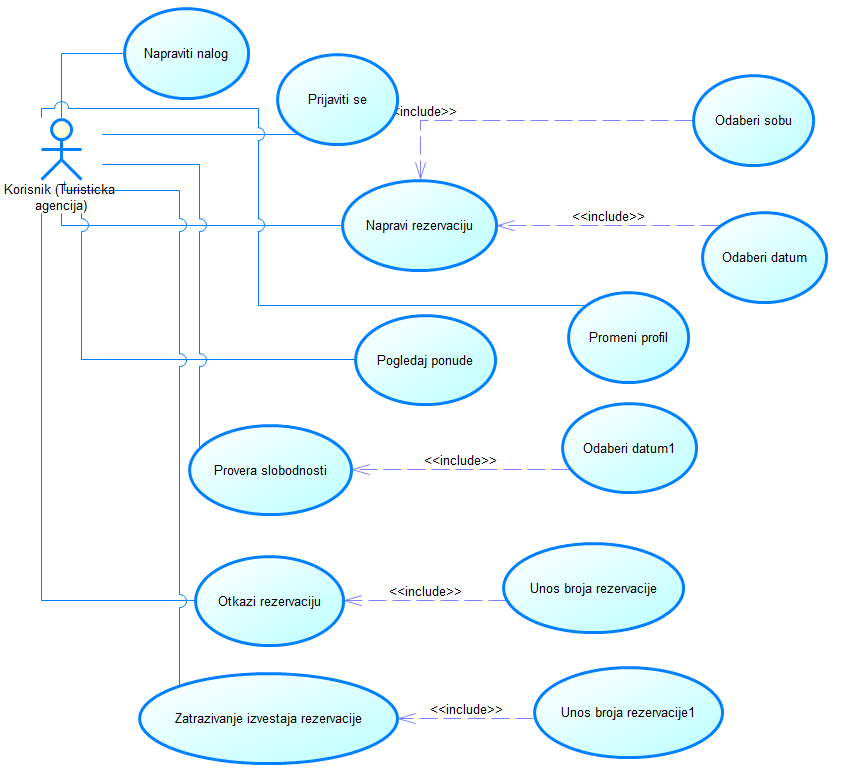
Slika 3.12.5. Predstavlja dijagram stanja

**Process View**

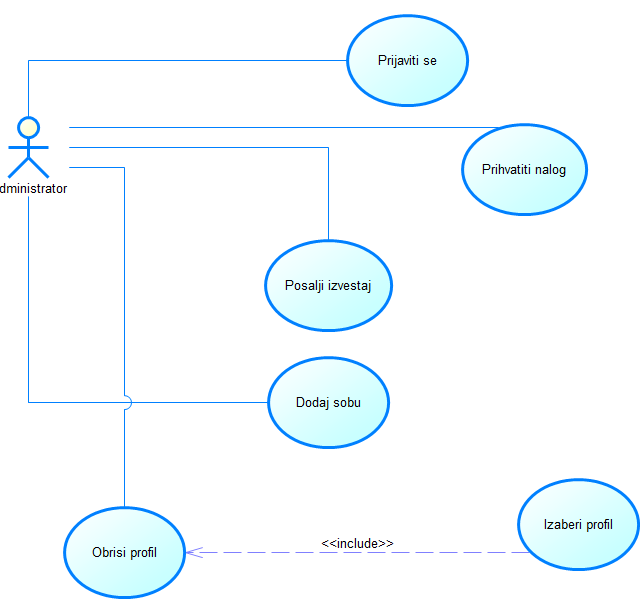
Ovaj razvoj se bavi dinamičkim aspektima sistema, tačnije objašnjava sistemske procese I kako oni komuniciraju i fokusiraju se na vremensko ponašanje sistema. Sam dijagram za ovaj razvoj je predstavljen u logic view i to je dijagram aktivnosti[4].

**Scenarios**

Opis arhitekture ilustrovan koristeći mali skup slučaja korišćenja ili scenarija, postaju peti pogled. Scenariji opisuju interakcije između objekta i procesa. Oni se koriste za indentifikaciju arhitektonske elemente i za ilustraciju i validaciju dizaj arhitekture. Sami dijagram u UML se nazivaju Use Case dijagrami.



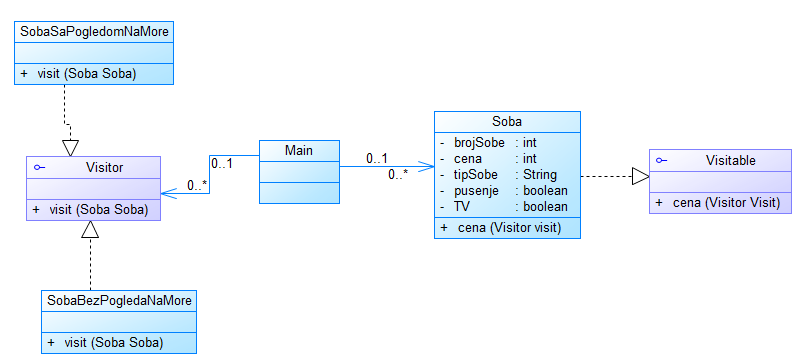
Slika 3.12.6. Use case za korisnika



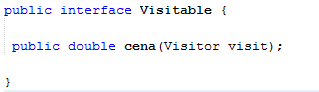
Slika 3.12.7. Use case za administratora

# 4. Upotreba uzorka Visitor

Na sledećoj slici možemo videti primer patterna Visitor. Kao što možemo videti na slici, imamo tri klase (ne računajući main klasu) i to su SobaSaPogledomNaMore, SobaBezPogledaNaMore i glavnu klasu Soba. Same klase sa tipom pogleda na more sadrže, tačnije implementiraju interferjs Visitor. Samom implementacijom ovog interfejsa kroz interfejs Visitable menja se cena sobe. U zavisnosti od tipa sobe koje je odabran, dobija se u Main klasi cena same sobe. Ukoliko je soba sa pogledom na more, cena sobe će biti skuplja, dok ukoliko je bez pogleda na more, cena sobe će biti niža.

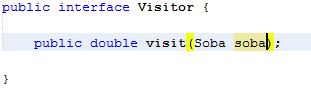


Slika 4.1. Predstavlja klasni dijagram klase Sobe koristeći pattern Visitor



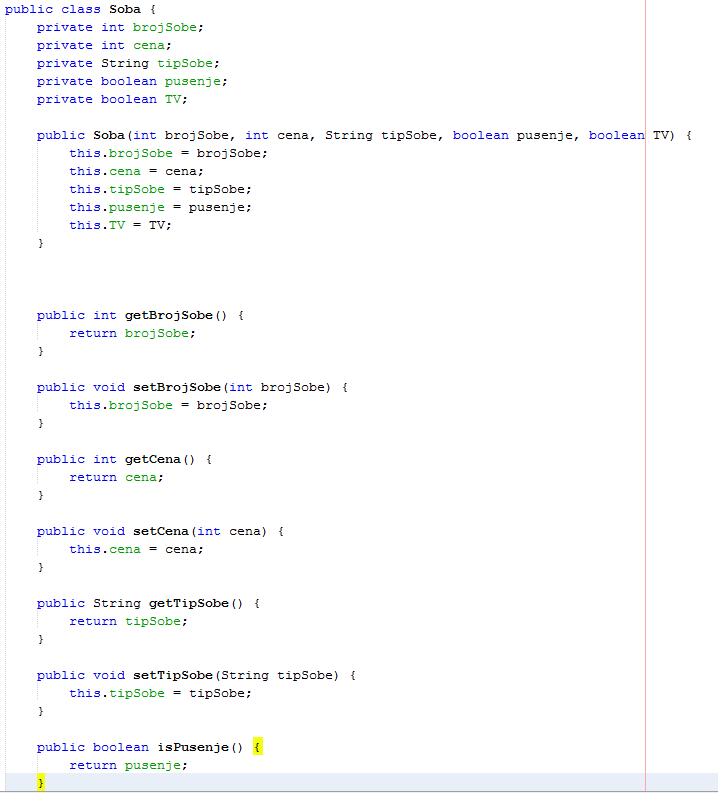
Slika 4.2. Interfejs Visitable

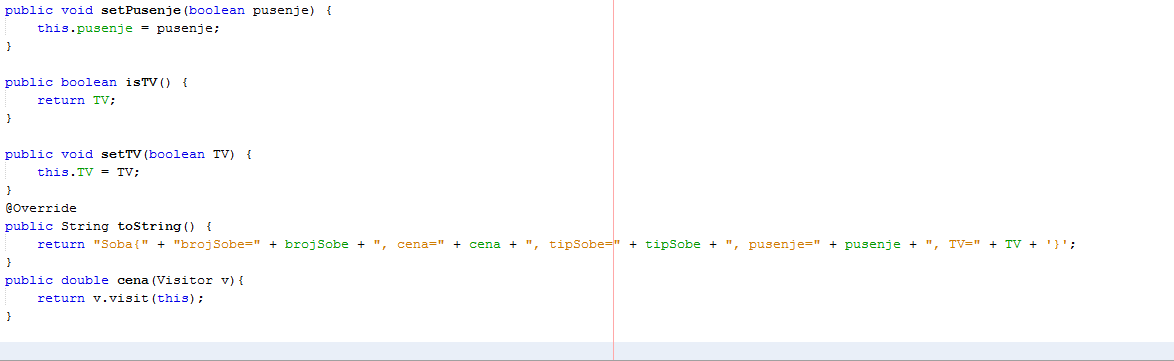
Na slici 4.2. možemo videti sam interfej Visitable. Kao što vidimo ovaj interfjs sadrži samo jednu metodu tipa double. Ta metoda je cena (ona se menja u zavisnosti o kojoj se sobi radi) i ona ima tu mogućnost da radi nad klasama koje implementiraju sam interfejs Visitor.



Slika 4.3. Interfjes Visitor

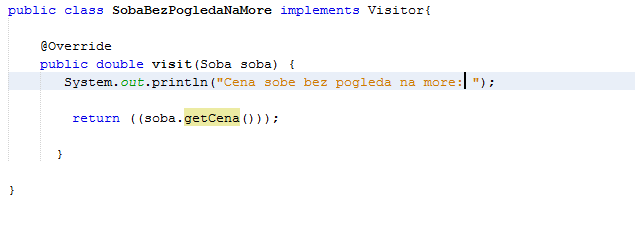
Slika 4.3. predstalja samu strukturu interfejsa Visitor. Kao što možemo primetiti, u ovom interfejsu takođe imamo samo jednu metodu koja je isto tipa double i to je metoda visit.





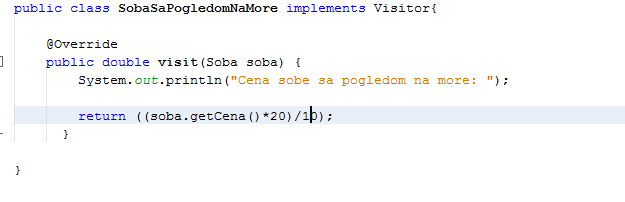
Slika 4.4. Klasa Soba

Na slici 4.4. Predstavljena je klasa soba. U ovoj klasi vidimo sve moguće atribute same klase (brojSobe, cena, tipSobe, pusenje, TV) zajedno sa konstruktorom koji sadrži sve atribute, isto tako imamo i get i set metode i samu toString metodu. Pored samog konstruktora i get/set metoda ova klasa sadrži metodu cena, koja će zavisiti od samog pogleda same sobe. Ukoliko soba ima pogled na more, cena sobe će biti već, dok ukoliko soba nema pogled na more, cena će biti manja.



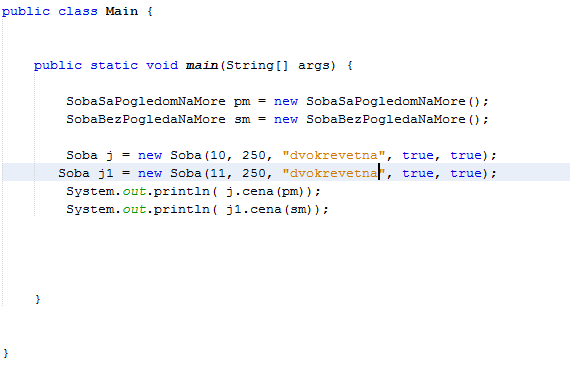
Slika 4.5. Klasa SobaBezPogledaNaMore

Na slici 4.5. možemo videti jedan od dva moguća pogleda iz sobe. Ova klasa se odnosi na sobu koja je jeftinija i nema pogled na more. Sama ova klasa implementuje interfejs Visitor i njenu abstraktnu metodu visit. Ova metoda ispusuje i vraća standardnu cenu sobe.



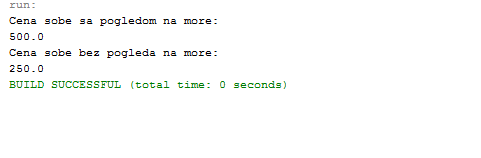
Slika 4.6. Klasa SobaSaPogledomNaMore

Na slici 4.6. možemo videti drugu mogućnost pogleda iz sobe. Ova klasa se odnosi na sobu koja je skuplja i ima pogled na more. Sama ova klasa implementuje interfejs Visitor i njenu abstraktnu metodu visit. Ova metoda ispusuje i vraća cenu sobe pomnođenu sa dvadeset i ceo taj iznos podeljen sa deset (računjanje sobe postavljen od strane vlasnika hotela).



Slika 4.7. Main klasa

Slika 4.7. predstavlja samu glavnu klasu. U ovoj klasi pozivamo naše klase (SobaSaPogledomNaMore i SobaBezPogledaNaMore). Nakon pozivanja pravimo instance klase Sobe i postavljamo sve potrebne atribute, koji su traženi u konstruktoru. Nakon toga prikazujemo putem j.cena(pm) – skuplja soba i j1.cena(sm) – jeftinija soba same cene sobe.



Slika 4.8. Krajni ishod aplikacije

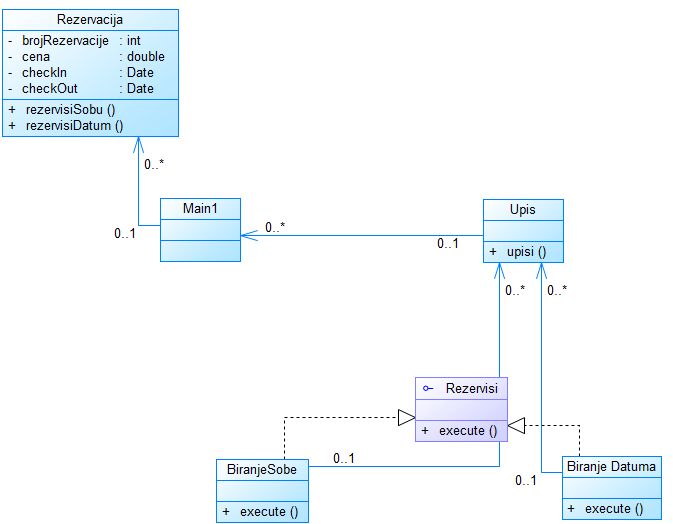
Na slici 4.8. možemo videti kako izgleda naš program nakon pokretanja.

**Prednosti i mane**

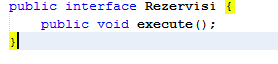
* Najveća prednost jeste dodavanje samih metoda i primene te metode. Kada dodamo metodu, mi nemamo potrebu da izvršavamo neku veću promenu, dok lako možemo da primenimo jednu metodu na svim klasama koje naravno pripadaju istoj hijerarhiji.
* Nedostatak samog ovog uzorka jeste što u svakoj klasi koja nasleđuju interfejs Visitable, mora da se pozove određena metoda iz tog interfejsa i što sama implementacija ovog uzorka, dovodi do potpune promene koda.

# 5. Upotreba uzorka Command

Pomoću ovog designe pattern-a uspećemo da pokažemo ispis rezervacije za odabrani datum i sobu. Na sledećoj slici videćemo da u ovom delu imamo jedan interfejs i klase koje implementiraju sam taj interfejs (a to su klasa BiranjeSobe i BiranjeDatuma). Naravno, pored toga tu je main klasa pomoću koje ćemo istestirati (izvršiti) nas program. Ove dve klase implementiraju nasleđenu funkciju tako što sadrže instancu klase Rezervacija i pozivaju njene metode rezervisiSobu () ili rezervisiDatum(). Sama klasa Upis, u zavisnosti od komande upisaju samu rezervaciju koju korisnik izabere. Sama sifra u klasi „Rezervacija“, predstaviće o kojoj sobi i kom datumu je reč. Na kraju ćemo videti i sam ishod, tačnije krajni rezultat aplikacije.

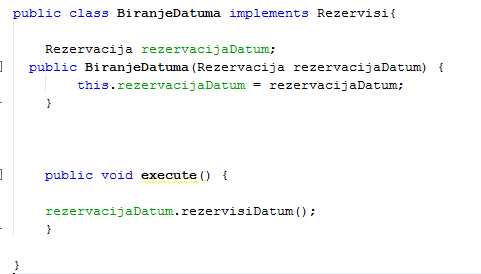


Slika 5.1 – Klasni dijagram koristeći command uzorka



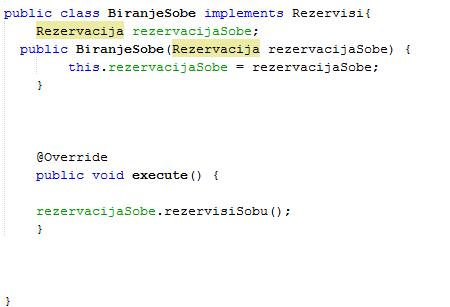
Slika 5.2. Interfejs rezervisi

Na slici 5.2. vidimo da imamo samo metodu execute, koja će obavljati određenu funkciju u zavisnosti od klase koja je nalsedila ovaj interfejs.



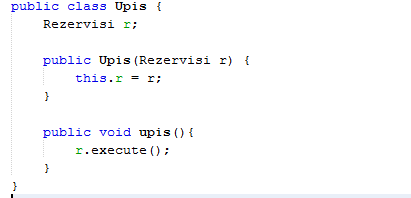
Slika 5.3. Klasa BiranjeDatuma

Slika 5.3. predstavtavlja jednu od dve klase koje implementiraju sam interfejs rezervisi. Osim same abstraktne metode, koja je dobijena implementacijom interfejsa i njeno postaljane (koja funkcija će biti obaljena), ova klasa sadrži i sam konstuktor sa postavljenim atrubitima iz klase rezervacija.



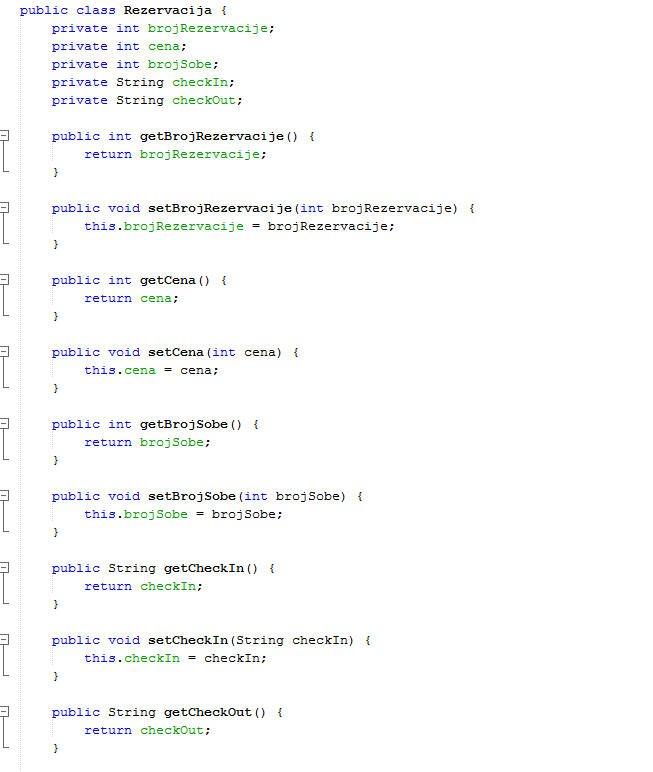
Slika 5.4. Klasa BiranjeSobe

Slika 5.4. predstavlja drugu klasu koja implementira sam interfejs rezervisi. Ova klasa isto kao i prethodna sadrži konstruktor i implementiranu metodu execute. Sama metoda se razlikuje, pošto se gore rezervisao datum ovde se rezerrviše soba.



Slika 5.5. Klasa Upis

Na slici 5.5. vidimo klasu upiši koja ima uljučenu (pozvanu klasu) rezervacija, konstruktor i samu metodu upis. Sama metoda nam pomaže da ispišemo sve date informacije od strane postavljenog konstruktora.





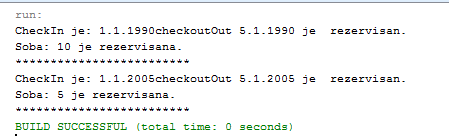
Slika 5.6. Klasa Rezervacija

Slika 5.6. predstavlja samu klasu Rezervacija, koja od atributa sadrži brojRezervacije, brojSobe, cenu, checkIn i checkout. Pored samih atributa, tu su konstruktori sa svim atributima a i prazni konstuktor, get/set metode i toString metoda. Poreh samih tih metoda, imamo i jako bitne metode. To su rezervisiSobu, koja nam pomaže da nakon postavljanja same rezervacije postavimo i ispišemo broj sobe, dok pomoću same metode rezervišiDatum, ispišemo datum same postavljene rezervacije.



Slika 5.7. Main klasa

Sama slika 5.7. predstalja main klasu. U njoj možemo da vidimo da kreiramo instance klase rezervacija i da nakon toga postavljamo same checkIn I checkout datume, pored toga vidimo (kako bi sama rezervacija bila potpuna mora se u rezervaciji sadržati i broj sobe) da i postavljamo sam broj sobe za željeni datum. Na kraju sve te funkcije ispisuje i izvršavamo putem klase Upis i njene metode upis (koju smo prethodno objasnili).



Slika 5.8. Krajni ishod aplikacije

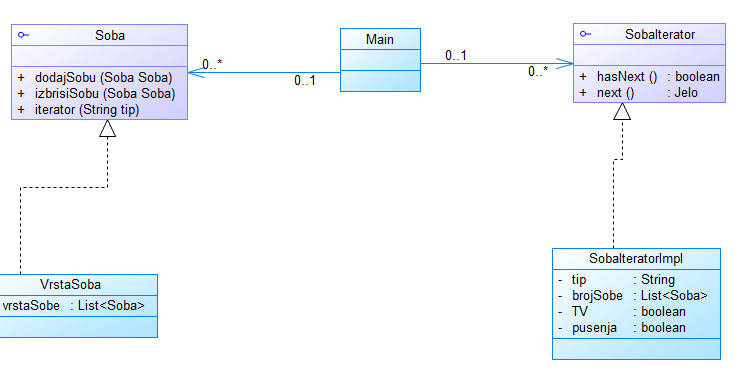
Na slici 5.8. možemo primetiti kako izgleda program nakon njegovog izvršavanja.

**Prednosti i nedostaci:**

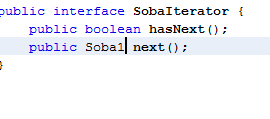
Prednosti ovog pattern-a je što postoji samo jedan metod (execute()) koji izvršava željene komande, takođe sama komanda može da se doda bez ikakve potrebe da se ceo kod menja i takođe ova komanda radi sa skrivenim objektom. Najveći nedostatak jeste što pravimo posebne klase za komande, što kasnije može napraviti problem.

# 6. Upotreba uzorka Iterator

U ovom projektu je uzorak iterator korišćen za prolazak kroz različite vrste soba. Kao što je prikazano, uzorak sadrži dva interfejsa i dve konkretne klase koje ih implementiraju. SobaIteratorImpl sadrži implementaciju metoda interfejsa SobaIterator i samim tim postaje konkretni iterator za bilo koji tip sobe.

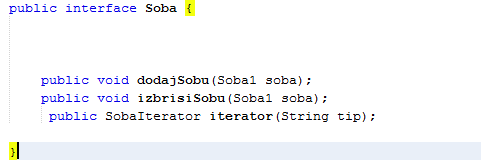


Slika 6.1 – Klasni dijagram primenjenog uzorka Iterator



Slika 6.2. Interfejs SobaIterator

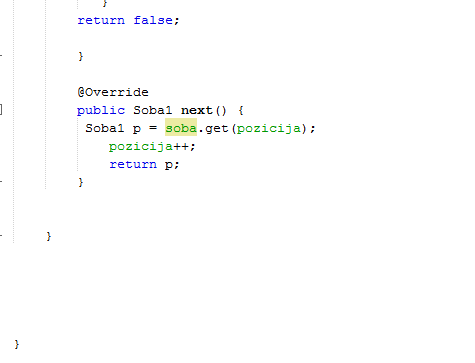
Na slici 6.2. možemo primetiti sam interfejs SobaIterator. Vidimo da on sadrži dve metode, koje će kasnije biti implementirane u neku od sledećih klasa.



Slika 6.3. Interfejs Soba

Na slici 6.3. možemo videti interfejs sobu, koja sadrži tri metode. Metoda dodajSobu nam omogućava da u listu soba dodamo novu sobu. Metoda izbrisiSoba nam ommogućava da iz liste soba, obrišemo određenu sobu. Metoda iterator će nam pomoći pri implementaciji samog iteratora.





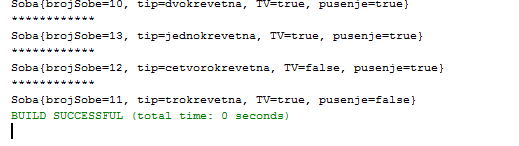
Slika 6.4. Klasa VrstaSoba

U ovoj klasi izvršićemo samu implementaciju samog interfejsa Soba. Kao što vidimo sve abstraktne metode smo implementirali i odredili im određenu funkciju. U ovoj klasi takođe vidimo i unutrašnju klasu SobaIteratorImplemenracija koja nasleđuje sam interfejs SobaIterator. Ova unutrašnja klasa implementira metode hasNext i next.



Slika 6.5. Main klasa

Na slici 6.5. možemo videti da smo postavili nekoliko tipova soba za pretragu (dvokrevetna, jednokrevetna…).Pre samog početka smo pomoću funkcije sveSobe, dodali sve specifikacije soba i nakon toga pustili pretragu po tipu sobe. Nakon izvršenja programa, naš program će izgledati ovako:



Slika 6.6. Krajni ishod aplikacije

**Prednosti i nedostaci uzorka Iterator**

Jako velika prednost samog ovog uzorka jeste, da kada napišemo jednu implementaciju, nju možemo koristiti za sve objekte. Nedostatak je jedini što se duplira posao, pošto same metode hasNext() i next () uzimaju samo poslednji element, pored toga stanje samog itertora bi bilo teško.

# 7. Mogućnosti i funkcije koje će biti testirane

Funkcije koje ćemo ovde testirati su:

* Podizanje sistema
* Pristup sistemu
* Provera ponuda
* Provera slobodnosti sobe
* Unos datuma rezervacije
* Biranje sobe
* Otkazivanje rezervacije

# 8. Mogućnosti i funkcije koje neće biti testirane

Same funkcije koje ćemo testirati u ovom projektu su softverskog karaktera. Sve funkcije koje ćemo testirati biće vezane za mogućnosti korisnika. Pored samih softverskih komponenti koje će biti testirane, trebalo bi testirati i hardverske. Same hardverske komponente će biti posmatrane kao da su ispravne i u ovoj fazi neće doći do njihovih testiranja.

# 9.Pristup/Strategija

Samo testiranje predstavlja integtalni deo ukupnog procesa razvoja. Testiranje našeg projekta biće zasnovano na testiranju na bazi modela.Samo testiranje će biti putem dve tehnologije koje smo obradili na samom predmetu SE321, a to su metode crne i bele kutije. Sama metoda bele kutije je metoda testiranja koja je slična metodi crne kutije. Tačnije samo testiranje aplikacije korišćenjem metode crne kutije, predstavlja testiranje samih funkcionalnosti sistema. To znači da tester testira svaku funkcionalnost i beleži dobijene rezultate. Nakon toga, vrši se poređenje sa postavljenim rezultatima koji su trebali da se dobiju. Sama ova tehnologija (tehnologija crne kutije) ne ulazi u samu strukturu koda. Kao što smo rekli bela kutija je metoda koja je slična crnoj kutiji, jedina razlika jeste to što ulazi u sam kod našeg sistema. Pored samih metoda crne i bele kutije u ovom projektu koristićemo i testiranje jedinica, integraciono testiranje, sistemsko testiranje, test projvatljivosti I instalaciono testiranje. Samo testiranje jedinica obuhvata testiranje jedinica koje su nezavisne. Ovo testiranje biće vršeno na podprograme uz pomoć debagovanja. Integraciono testiranje ima za cilj da nakon jediničnog testiranja, integriše i zatim pronađe grešku i nakon toga da se te greške uklone. Kod sistemskog testiranja, glavna stvar koja se testira jesu nefunkcionlin zahtevi samog sistema. To se odnosi na bezbednost, brzinu, preciznost ili pouzdanost. Testiranje prihvatiljovisti se odnosi na testiranje od strane korisnika.Tačnije korisnik podnosi jedan od prethodnih zahteva i na taj način provera da li su zahtevi ispunjeni. Kao poslednje testiranje jeste instalaciono. Samo ovo testiranje se odnosi kako će se sistem ponašati prema hardverkoj konfiguraciji.

# 10. Kriterijumi za ulaz/izlaz iz faze testiranja

Kako bi uopšte testirali neki modul, sam modul mora zadovoljiti neke od kriterijuma:

* Kod mora biti pravilno kompajliran
* Svi testovi razvojnog tima moraju biti uspešno izršeni
* Korisnički zahtevi se ne smeju menjati

# 11. SUT Prošao/Nije prošao kriterijumi

U ovom delu prikazaćemo tačnije postavićemo kriterijume za broj grešaka koje se otkrivaju u trenutku testiranja I njihov prioritet.

**12. Kriterijum obustavljanja testiranja i potrebni preduslovi za nastavak**

Samo obustavljanje testiranja je moguće ukoliko dođe do:

* Pojave greške 1, 2 ili 3 kategorije
* Sistem ne obrađuje ulazne podatke
* Greške u hardveru

Ukoliko dođe do defekta, koji će prekuniti rad softvera, mora se praviti nova verzija sistema koja mora prođi kroći kroz sce regresivne testove. Ukoliko svi ti testovi prođu uspešno, samo testiranje će se nastaviti po originalnom planu. Videli smo da je moguće da dođe do otkaza hardvera, u tom slučaju sam tester mora obavestiti ljude koji su zaduženi za to. Nakon popravke hardvera, ponovno testiranje je opet moguće.

**13. Potrebno osoblje i obuka**

Pošto je sam softver i period izrade na početku određen i ograničen, sam procestestiranja će biti zasnivan na aplikacijama koje su testeri već koristili. Sam tim za testiranje će se sastojati od QA menadžera, menadžera testera i samih testera. Samo testiranje biće uloženo puno radon vreme i svi članovi tima će ga poštovati, jedini izuzetak je ukoliko traba da dođe do dodatne radne snage i njihovo dovođenje i radon vreme će odlučivati menažer.

. **14. Odgovornosti i održavanje**

|  |  |  |  |
| --- | --- | --- | --- |
| R.B. | Pozicija i zaduženje | Ime i prezime | Kontakt informacije |
| 1. | Menadžer projekta- radi samu organizaciju, sastavlja tim, vodi računa o dokumentaciji projekta. | Ivan Blagojević | 123/123/123 Nnn @nnn.com |
| 2. | QA menadžer – vodiće detaljnu dokumentaciju o proizvodu, koje su performance, koja je namena i koje su pronađene greške i propusti | Marko Nikolić | 123/123/123 Nnn @nnn.com |
| 3. | Test menadžer – nadgleda rad testera, daje predloge mogućih testova i beleži sve pronađene greške koje su pronađene | Nikola Jović | 123/123/123 Nnn @nnn.com |
| 4. | Tester – obavlja testove i beleži dobijene rezultate | Saša Vidanović | 123/123/123 Nnn @nnn.com |
| 5. | Tester – obavlja testove i beleži dobijene rezultate | Marko Ilić | 123/123/123 Nnn @nnn.com |
| 6. | Tester – obavlja testove i beleži dobijene rezultate | Ilija Popović | 123/123/123 Nnn @nnn.com |

**15. Vremenski raspored**

|  |  |  |  |
| --- | --- | --- | --- |
| Faza | Broj iteracije | Početak | Kraj |
| Početna | 1 | Nedelja#1 | Nedelja#4 |
| Faza kompletiranja zahteva | 2 | Nedelja#4 | Nedelja#7 |
| Faza implementacije | 3 | Nedelja#7 | Nedelja#18 |

Na prethodnoj tabeli možemo videti, planirane iteracije i faze samog razvoj projekta. Kao što možemo videti sama početna faza trajaće četiri nedelje, sama faza za kompletiranje zahteva spade kao 2 iteracija koja će traja tri nedelje. Kao krajna faza, biće faza implementacije koja će trajati jedanaest nedelja.

Na sledećoj tabeli videćemo deljniji prikaz prethodne tabele:

|  |  |  |  |
| --- | --- | --- | --- |
| Faza | | | Početak/Kraj |
| Početna faza | Iteracija 1 | Biznis model | Početak: 22.1.2017.  Kraj: 19.2.2017. |
| Analiza zahteva |
| Konfiguracija |
| Faza kolaboracije | Iteracija 2 – Razvoj prototipa arhitekture | Biznis model | Početak: 19.2.2017.  Kraj: 12.3.2017. |
| Tehnički zahtevi |
| Analiza i dizajn |
| Implementacija |
| Testiranje komponenti |
| Faza konstrukcije | Interacija 3.1.  Release 1 | Implementacija Beta verzije | Početak:12.3.2017.  Kraj:28.5.2017. |
| Testiranje |
| Inteacija 3.2.  Release 2 | Analiza i dizajn |
| Implementacija |
| Integralno testiranje |
| Interacija 3.3.  Release 2.1 | Analiza i dizajn |
| Implementacija |
| Test prihvatljivosti |

1. **Test okruženje**

Samo test okruženje činiće:

Procesor : Intel (R) Core(TM) i3 – 2100 3.1 GHz

RAM: 8GB

System type: 32-bit operating system, x86-based processor

Server

Procesor: Intel(R) Xeon(R) CPU E5-2630 v3, 32-Core Server Processor

RAM: 128GB DDR4 RAM

HDD: 100GB

**17. Budžet i troškovi**

Sam budžet koji će biti potreban za ovaj projekta biće određen korišćenjem COCOMO modela.

Prvo moramo odrediti o kojoj se vrsti projekta radi. Sledeća tabela će prikazati sve moguće tipove softera (u zavisnosti od broja linija koda):

|  |  |  |
| --- | --- | --- |
| Model | A | b |
| Oganic (2-50 KLOC), manji tim,  stabilni, bez mnogo iteracija | 2.4 | 1.05 |
| Semi- detached (50-300 KLOC),  srednji tim, prosečne pogodnosti, srednje vremenski ograničen | 3.0 | 1.12 |
| Embedded (>300), veći projekti,  kompleksni, inovativni, sa  ozbiljnim ograničenjima | 3.6 | 1.20 |

Naša aplikacija ima 16000 linija koda, tacnije 16KLOC.

Sam napor računaćemo po formulu E = a (KLOC) ^b , dok ćemo samo trajanje projekta računati putem sledeće formule TDEV = c (E) ^ d (c i d su prikazani u sledećoj tabeli zavise od vrsta moddela ), samu produktivnost ćemo računati putem formule P= size (LOC)/E a potrebno osoblje ćemo računati putem

formule SS= E/TDEV .

|  |  |  |
| --- | --- | --- |
| Model | C | d |
| Oganic (2-50 KLOC), manji tim,  stabilni, bez mnogo iteracija | 2.5 | 0.38 |
| Semi- detached (50-300 KLOC),  srednji tim, prosečne pogodnosti, srednje vremenski ograničen | 2.5 | 0.35 |
| Embedded (>300), veći projekti,  kompleksni, inovativni, sa  ozbiljnim ograničenjima | 2.5 | 0.32 |

Napor = E = 2.4 \* (16) ^1.05 = 44 ljudi-mesec

Trajanje = TDEV = 2.5 \*(44) ^ 0.38 = 11 meseci

Produktivnost = P= size (LOC)/E = 16000/44 = 363 LOC/ ljudi-mesec

Potrebno osoblje = SS = E/TDEV = 44/11= 4 osoblja

|  |  |  |  |
| --- | --- | --- | --- |
| Napor | Trajanje | Produktivnost | Potrebno osoblje |
| 44 ljudi-mesec | 11 meseci | 363 LOC/ljudi-mesec | 4 osoblja |

Uz sam Cocomo model prikazaćemo i troškove do kojih će doći. Na sledećoj tabeli prikazaćemo tabelu troškova za ljudske resurse.

|  |  |  |  |
| --- | --- | --- | --- |
| R.B. | Pozicija i zaduženje | Broj sati | Satnica |
| 1. | Menadžer projekta- radi samu organizaciju, sastavlja tim, vodi računa o dokumentaciji projekta. | 80sati | 55e |
| 2. | QA menadžer – vodiće detaljnu dokumentaciju o proizvodu, koje su performance, koja je namena i koje su pronađene greške i propusti | 72sata | 50e |
| 3. | Test menadžer – nadgleda rad testera, daje predloge mogućih testova i beleži sve pronađene greške koje su pronađene | 56sati | 40e |
| 4. | Tester – obavlja testove i beleži dobijene rezultate | 40sati | 25e |
| 5. | Tester – obavlja testove i beleži dobijene rezultate | 40sati | 25e |
| 6. | Tester – obavlja testove i beleži dobijene rezultate | 40sati | 25e |
| 7. | Projektni tim | 48 sati | 25e/po osobi |
| 8. | Programer | 32sata | 15e |
| 9. | Administrator baze | 24sata | 10e |

Na same ljudske resurse potrebno je uložiti dosta. Samim računanjen cena koja je potrebna za ljudske resurse jeste:

* Menadžer projekta = 4.400e
* QA menadžer = 3.600e
* Test menadžer = 2.240e
* Tester x3 =3.000e
* Projektni tim = sastoji se od 7 ljudi = 8.400e
* Programerx2 = 960e
* Administrator baze = 240e

Ukupni troškovi za ljudske resurse su 22,840e. Naravno ovo je samo potrebno sredstvo za same radnike, tu su naravno uključeni ostali faktori i faze koje zahtevaju svoj budžet. Budžet za same faze jeste 20.000e. Na kraju možemo zaključiti da ukupan budžet za ceo projekat biće 45.000e, u ovo cenu uključilo smo i dodatan trošak koji je moguće da nastane u toku rizika (rizici prikazani u sledećem poglavlju).

# 18.Rizici sistema

U ovom delu su navedeni svi identifikovani rizici koji mogu ugroziti realizaciju projekta, kao i predlozi kako delovati u slučaju njihovog pojavljivanja (**CONTINGENCIE PLAN**):

|  |  |
| --- | --- |
| RIZIK | **CONTINGENCIE PLAN** |
| Nedostatak alata | Ukoliko tim koji je zadužen za ovo, ne dostavi alate, samo testiranje će morati da se odloži i moguće je doći do angažovanja još potrebnih radnika. |
| Veliki broj grešaka | Ukoliko dođe do ovoga, menadžer mora da se sastane sa vođom projekta i odluči o daljem razvijanju projekta. Najbolji slučaj je da se odloži sama implementacija ove verzije, dok se greške ne smanje. |
| Nedostatak vremena za izradu testova | Ukoliko dođe do ovoga, potrebno je produžiti rok ili u najgorem slučaju izbacujemo testove koji su manjeg prioriteta. |
| Nedostatak test osoblja | Potrebno je angažovati dodatno osoblje ili odustati od nekih testova. |

# 19.Testiranje

Samo testiranje će biti iz dve faze, prva faza jeste sam plan testiranja. U njemu ćemo postaviti koji je use case, koja se funkcija testira, koje je stanje sistema, koji su ulazni podaci i na kraju koji je ishod. Nakon predstavljanja plana testiranja, prikazaćemo samo testiranje.

## 19.1 Plan testiranja

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Use Case | Function Being Tester | Initial System State | Input | Expected output |
| Podizanje sistema | Testira se samo pokretanje funkcije | Sistem se pokreće | Kuca se stranica aplikacije | Sistem je uspešno pokrenut |
| Pristup sistemu | Testira se sama login forma | Sistem je pokrenut | Unosi se korisničko ime i šifra + enter | Sistemu je uspešno pristupljeno |
| Provera svih soba | Testira se sama funkcija prikazivanja ponuda hotela | Sistem je pokrenut | Bira se opcija prikaz soba | Ponude su prikazane |
| Provera slobodnosti sobe | Testira se sama provera slobodnosti sobe | Proverene sobe | Unosi se datum za proveru + enter | Dostavljan je broj slobodnih soba |
| Unos datuma rezervacije | Testira se opcija biranje datuma | Odabran datum | Unosi se datum rezervacije + enter | Odabran je datum rezervacije |
| Biranje sobe | Testira se opcija biranje sobe | Odabrana soba | Unosi se broj sobe  +enter | Odabran je broj sobe |
| Otkazivanje rezervacije | Testira se sama opcija za otkazivanje aplikacije | Otkazana rezervacija | Unosi se broj rezervacije  + enter | Rezervacije je otkazana. |

## 19.2. Samo testiranje

**Pristup sistemu**

prijava(korisnickoIme, sifra), proveraPodataka(), getNalog()

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Naslov | Pristup sistemu | Rev | 1 | Autor | Ivan Blagojević | | Datum | 22.1.2017 |
| Cilj | Provera unetih podataka | Reference |  |  |  | |  |  |
| Test uslovi | Vreme neophodno za izradu test slučaja | 20min | Vreme neophodno za izvršenje test slučaja | | | 5min | | |

|  |  |
| --- | --- |
| Opis postavke za testiranje | |
| 1. | Korisnik je uneo korisničko ime i šifru |
| 2. | Korisnik je potvrdio unos podataka |
| 3. | Korisnik dobija poruku da je pogrešno uneo korisničke podatke |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Definicija testa Izvršenje testa | | | | | |
|  | Uslovi | Ulazni podaci | Očekivani rezultati | Aktuelni rezultati | Broj problema |
|  | Korisnik pristupa sistemu pomoću login forme | Korisnik unosi korisničko ime i šifru | Uspešno pristupanje sistemu | Korisnik nije uspešno pristupio sistemu |  |

|  |  |
| --- | --- |
| Opis postuslova | |
| 1. | Korisnik je uspešno uneo korisničke podatke i pristupio sistemu. |

**Pregled svih soba**

prijava(korisnickoIme, sifra), getNalog() , uvidInfoSobe()

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Naslov | Provera ponuda | Rev | 1 | Autor | Ivan Blagojević | | Datum | 22.1.2017 |
| Cilj | Provera prikazivanja ponuda hotela | Reference |  |  |  | |  |  |
| Test uslovi | Vreme neophodno za izradu test slučaja | 20min | Vreme neophodno za izvršenje test slučaja | | | 5min | | |

|  |  |
| --- | --- |
| Opis postavke za testiranje | |
| 1. | Korisnik je odabrao opciju “Prikaži ponude” |
| 2. | Korisnik dobija poruku da nema ponuda (razlog toga može biti da nema pristup internet konekciji, mogu biti problem sa bazom, nije ulogovan na sistem) |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Definicija testa Izvršenje testa | | | | | |
|  | Uslovi | Ulazni podaci | Očekivani rezultati | Aktuelni rezultati | Broj problema |
|  | Korisnik ima mogućnost prikazivanja ponuda hotela. | Korisnik bira opciju “Prikaži ponude” | Uspešno prikazane ponude hotela | Korisnik nije odabrao opciju za prikaz ponuda |  |

|  |  |
| --- | --- |
| Opis postuslova | |
| 1. | Korisnik je uspešno odabrao opciju prikaz opcija i dobio je listu svih ponuda samog hotela |

**Provera slobodnosti sobe**

prijava(korisnickoIme, sifra), getNalog(), uvidUInfoSobe()

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Naslov | Provera slobodnosti sobe | Rev | 1 | Autor | Ivan Blagojević | | Datum | 22.1.2017 |
| Cilj | Provera slobodnosti sobe | Reference |  |  |  | |  |  |
| Test uslovi | Vreme neophodno za izradu test slučaja | 20min | Vreme neophodno za izvršenje test slučaja | | | 5min | | |

|  |  |
| --- | --- |
| Opis postavke za testiranje | |
| 1. | Korisnik je uneo datum za koji želi da vidim slobodnost sobe. |
| 2. | Korisnik je potvrdio unos podataka. |
| 3. | Korisnik dobija poruku da je uneo invalidan datum (uneo datum koji je već prošao) |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Definicija testa Izvršenje testa | | | | | |
|  | Uslovi | Ulazni podaci | Očekivani rezultati | Aktuelni rezultati | Broj problema |
|  | Korisnik ima mogućnost provere slobodnosti sobe | Korisnik unosi datum | Uspešno prikazan broj slobodnih soba | Korisnik je uneo invalidan datum (uneo je datum koji je već prošao) |  |

|  |  |
| --- | --- |
| Opis postuslova | |
| 1. | Korisnik je uspešno uneo datum i dobio broj slobodnih soba za taj datum |

**Unos datuma rezervacije**

prijava(korisnickoIme, sifra), getNalog() , rezervacija(), biranjeDatuma(), setDatum(), provera().

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Naslov | Unos datuma rezervacije | Rev | 1 | Autor | Ivan Blagojević | | Datum | 8.1.2017 |
| Cilj | Provera odabira datuma za rezervaciju | Reference |  |  |  | |  |  |
| Test uslovi | Vreme neophodno za izradu test slučaja | 20min | Vreme neophodno za izvršenje test slučaja | | | 5min | | |

|  |  |
| --- | --- |
| Opis postavke za testiranje | |
| 1. | Korisnik je uneo datum rezervacije. |
| 2. | Korisnik je potvrdio unos podataka. |
| 3. | Korisnik dobija poruku da je odabrao invalidni datum. |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Definicija testa Izvršenje testa | | | | | |
|  | Uslovi | Ulazni podaci | Očekivani rezultati | Aktuelni rezultati | Broj problema |
|  | Korisnik ima mogućnost biranja datuma razervacije | Korisnik unosi datum | Uspešno odabran datum rezervacije | Korisnik nije uspešno odabrao datum rezervacije |  |

|  |  |
| --- | --- |
| Opis postuslova | |
| 1. | Korisnik je uspešno izabrao datum rezervacije. |

**Biranje sobe**

prijava(korisnickoIme, sifra), getNalog() , rezervacija(), biranjeSobe(), setBrojSobe(), provera()

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Naslov | Biranje sobe | Rev | 1 | Autor | Ivan Blagojević | | Datum | 22.1.2017 |
| Cilj | Provera odabira broja sobe | Reference |  |  |  | |  |  |
| Test uslovi | Vreme neophodno za izradu test slučaja | 20min | Vreme neophodno za izvršenje test slučaja | | | 5min | | |

|  |  |
| --- | --- |
| Opis postavke za testiranje | |
| 1. | Korisnik je uneo broj sobe. |
| 2. | Korisnik je potvrdio unos podataka |
| 3. | Korisnik dobija poruku da je pogrešno uneo broj sobe. |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Definicija testa Izvršenje testa | | | | | |
|  | Uslovi | Ulazni podaci | Očekivani rezultati | Aktuelni rezultati | Broj problema |
|  | Korisnik ima mogućnost da unese broj sobe. | Korisnik unosi broj sobe. | Uspešno unet broj sobe. | Korisnik nije uspešno uneo broj sobe. |  |

|  |  |
| --- | --- |
| Opis postuslova | |
| 1. | Korisnik je uspešno uneo broj sobe. |

**Otkazivanje rezervacije**

prijava(korisnickoIme,sifra), getNalog() , unosBrojaRezervacije(), setBrojRezervacije(), otazRezervacije()

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Naslov | Otkazivanje rezervacije | Rev | 1 | Autor | Ivan Blagojević | | Datum | 22.1.2017 |
| Cilj | Provera otkazivanja rezervacije | Reference |  |  |  | |  |  |
| Test uslovi | Vreme neophodno za izradu test slučaja | 20min | Vreme neophodno za izvršenje test slučaja | | | 5min | | |

|  |  |
| --- | --- |
| Opis postavke za testiranje | |
| 1. | Korisnik je uneo šifru rezervacije. |
| 2. | Korisnik je potvrdio unos podataka. |
| 3. | Korisnik dobija poruku da je pogrešno uneo šifru rezervacije. |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Definicija testa Izvršenje testa | | | | | |
|  | Uslovi | Ulazni podaci | Očekivani rezultati | Aktuelni rezultati | Broj problema |
|  | Korisnik ima mogućnost da otkaže rezervaciju | Korisnik šifru rezervacije | Uspešno otkazivanje rezervacije | Korisnik nije uspešno otkazao rezervaciju |  |

|  |  |
| --- | --- |
| Opis postuslova | |
| 1. | Korisnik je uspešno uneo šifru rezervacije i otkazao rezervaciju. |

# 20. Metrika

Metrika je kvantitivna mera stepena do koje je mere sistem, same sistemske komponente, ili process izvršava date attribute [6]. U sledećoj tabeli prikazaćemo analize od testera, koji su bili uključeni u fazi testiranja:

|  |  |  |
| --- | --- | --- |
| Broj | Test | Podaci dobijeni tokom test slučaja i njihovih izvršavanja |
| 1 | Broj zahteva | 11 |
| 2 | Prosečni broj test slučaja po zahtevu | 15 |
| 3 | Ukupna broj test slučaja napisanih za sve zahteve | 165 |
| 4 | Broj odrađenih test slučaja | 150 |
| 5 | Broj uspešnih testova | 130 |
| 6 | Broj neuspešnih testova | 20 |
| 9 | Ukupan broj grešaka | 25 |
| 10 | Broj grešaka nižeg prioriteta | 15 |
| 11 | Broj srednjeg prioriteta | 7 |
| 12 | Broj visokog prioriteta | 3 |

Pomoću formule (broj zspešnih testora/ukupan broj testova) možemo izračunati procenat uspešnosti, za naš sistem nakon samog završetka testiranja imamo rezulat:

Procenat uspešnosti = 130/165 \*100 = 78%

Pored samog procenta uspešnosti, možemo izračunati i procenat neuspešnosti. U našem sistemu taj procenat je jednak:

Procenat neuspešnosti = 20 /165 \* 100 = 12%

Slika 20.1. Dijagram test slučajeva

Isto tako od velikog značaja je da izračunama Defect Density. Njega računamo po formuli:

lg=Big/Bz

lg – intezitet grešaka

Big – broj identifikovanih grešaka

Bz – broj zahteva

lg =

𝒍𝒈 = 25/11 = 2.2

Kao zaključak dobijamo da po zahtevi pojavljivanje moguće greške je 2.2.

Sledeća jako bitna stvarka jeste broj grešaka po prioritetu. Same greške delimo na 3 tipa:

* Greške nižeg prioriteta
* Greške srednjeg prioriteta
* Greške visokog prioriteta

Sam procenat tih grešaka računamo po sledećoj formuli:

P = greške n prioriteta / broj grešaka \* 100

Greške nižeg reda = 15/25 \*100 = 60%

Greške srednjeg reda = 7/25 \*100 = 28%

Greške visokog reda = 3/25 \*100 = 12%

Slika 20.2. Dijagram grešaka po prioritetu

**Literatura:**

1. "The Zachman Framework: The Official Concise Definition" Zachman International. 2008.
2. *A Comparison of the Top Four Enterprise Architecture Methodologies*, Roger Sessions, Microsoft Developer Network Architecture Center
3. Kruchten, Philippe (1995, November). Architectural Blueprints — The “4+1” View Model of Software Architecture. IEEE Software 12 (6), pp. 42-50.
4. Mikko Kontio (2008, July) Architectural manifesto: Designing software architectures, Part 5
5. Dr. Ljubomir Lazic, MSc Nebojša Gavrilović, Projektovanje i Arhitektura softvera (2016)
6. Important Software Test Metrics and Measurements – Explained with Examples and Graphs, <http://searchsoftwarequality.techtarget.com/guides/Quality-metrics-A-guide-to-measuring-software-quality>
7. Slobodan Jovanović. Konstruisanje Softvera (Prvo izdanje), Beograd 2014.