UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Andrej Ban

RAZVOJ SPLETNE APLIKACIJE ZA NAJEM APARTMAJEV PO METODOLOGIJI RUP

DIPLOMSKO DELO NA
VISOKOŠOLSKEM STROKOVNEM ŠTUDIJU

Mentor: viš. pred. dr. Igor Rožanc

Ljubljana, 2009

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00427/2009
Datum: 15.02.2009

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **ANDREJ BAN**

Naslov: **RAZVOJ SPLETNE APLIKACIJE ZA NAJEM APARTMAJEV PO METODOLOGIJI RUP**

THE DEVELOPMENT OF APARTMENT RENTAL WEB APPLICATION USING RUP METHODOLOGY

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija

Tematika naloge:

Nekateri pristopi pri razvoju spletnih tehnologij omogočajo učinkovito izdelavo tipičnih poslovnih spletnih aplikacij, ki razvijalcu precej olajšajo delo.

V diplomski nalogi prikažite tipičen razvoj javanske spletne aplikacije z uporabo ogrodja Struts in strežniških javanskih zrn. V ta namen sistematično predstavite razvoj aplikacije za najem apartmajev v planinskih kočah, ki ima več vrst uporabnikov, podatke pa hrani v oraclovi podatkovni zbirki. Analizo zahtev, načrtovanje in konstrukcijo izvedite v skladu z zahtevami metodologije RUP. Nalogo zaključite s prikazom uporabe razvite aplikacije za različne uporabnike.

Mentor:

viš. pred. dr. Igor Rožanc

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:

prof. dr. Franc Solina

I Z J A V A O A V T O R S T V U

diplomskega dela

Spodaj podpisani/-a ANDREJ BAN,

z vpisno številko 63010180,

sem avtor/-ica diplomskega dela z naslovom:

Razvoj spletne aplikacije za najem apartmajev po metodologiji RUP

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal/-a samostojno pod mentorstvom (naziv, ime in priimek)

  viš. pred. dr. Igorja Rožanca

  in somentorstvom (naziv, ime in priimek)

  _____

- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela

- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne _____ Podpis avtorja/-ice: _____

Zahvala

Svojemu mentorju dr. Igorju Rožancu se iskreno zahvaljujem za vso podporo, nasvete in praktično pomoč.

Za potrpežljivost in vzpodbudo se zahvaljujem svojim staršem, ki so mi omogočili študij in me vsa leta podpirali.

Kazalo

Uporabljene kratice

API	Programski vmesnik (Application Programming Interface)
CORBA	Arhitektura, ki omogoča komunikacijo med objekti ne glede na programski jezik in operacijski sistem (Common Object Request Broker Architecture)
EAR	Strežniška arhivska datoteka (Enterprise ARchive)
EJB	Strežniška Javanska zrna (Enterprise Java Beans)
HTML	Programski jezik za ustvarjanje spletnih strani in drugih dokumentov (Hyper Text Markup Language)
HTTP	HTTP protokol (Hyper Text Transfer Protocol)
IDE	Integrirano razvojno okolje (Integrated Development Environment)
JAR	Javanska arhivska datoteka (Java ARchive)
Java SE	Standardna različica javanske platforme (Java Platform, Standard Edition)
Java EE	Podjetniška različica javanske platforme (Java Platform, Enterprise Edition)
JDBC	Standarden programski vmesnik za dostop do podatkovnih baz v Javi (Java Database Connectivity)
JNDI	Standardni vmesnik za lociranje storitev v Javi (Java Naming and Directory Interface)
JPA	Javanski programski vmesnik za delo s trajnimi objekti (Java Persistence API)
JSP	Javanske strežniške strani (Java Server Pages)
JVM	Javansko izvajalno okolje (Java Virtual Machine)
MVC	Model-pogled-krmilnik (Model View Controller)
RUP	Ogrodje objektnega procesnega modela (Rational Unified Process)
RMI	Javin model za oddaljeno proženje metod (Remote Method Invocation)
RMI-IIOP	Javin model za oddaljeno proženje metod (skladno z CORBA določili) (Remote Method Invocation over Internet Inter-Orb Protocol)
UML	Jezik za modeliranje (Unified Modeling Language)
WAR	Spletna arhivska datoteka (Web ARchive)

Povzetek

Diplomsko delo opisuje izdelavo spletne aplikacije z uporabo javanske platforme, tehnologije Struts in strežniških Javanskih zrn (EJB).

V diplomskem delu je podrobneje predstavljen primer izdelave spletne aplikacije za rezervacijo smučarskega apartmaja v eni izmed ponujenih koč.

V uvodnem poglavju je na kratko predstavljena razlaga razvoja spletne aplikacije za rezervacijo smučarskega apartmaja.

Sledi predstavitev vseh orodij in tehnologij, ki smo jih uporabljali pri načrtovanju in razvoju omenjene spletne aplikacije. Vsako orodje in tehnologija sta na kratko opisana, predstavljene pa so tudi njihove lastnosti.

Osrednji del diplome je predstavitev poteka analize in načrtovanja ter razvoja spletne aplikacije. Analiza, načrtovanje in razvoj spletne aplikacije so izvedeni z uporabo metodologije RUP, ki zajema štiri faze življenjskega cikla. Prva faza obsega primer uporabe, ki opisuje, kaj je ključnega pomena za razumevanje zahtev. Sledi predstavitev podatkovnega in procesnega dela, v katerem realiziramo različne tehnične diagrame. V nadaljevanju sledi konstrukcija sistema po načrtu iz prejšnjih faz. Zadnja faza pa predstavlja namestitev in testiranje aplikacije.

Četrto poglavje opisuje predstavitev in uporabo delovanja spletne aplikacije, kot jo vidi uporabnik.

Ključne besede:

- Java
- spletna aplikacija
- trinivojska arhitektura
- JSP
- EJB
- Struts
- RUP

Abstract

Diploma contains production of web application using the Java platform, Struts technology and Enterprise Java Beans.

In the diploma a case of web application for the reservation of ski apartments development is presented.

First chapter contains a brief explanation of the development web application for the reservation of ski apartments.

This chapter is followed with a presentation of all tools and technologies used for design and development of this web application. Each tools and technology is briefly described, and their properties are presented as well.

The main part of diploma contains analysis and design of particular web application development using Rational Unified Process methodology, which contains four phases of life cycle. The first phase is about the understanding of the requirements, while next presents the design of data and process view, which we realize by different diagrams. The third phase is about construction of the system according to the plan from the previous phase. The last phase presents the installation and testing of application.

The fourth chapter demonstrate the use and operation of the web application as the user sees it.

Keywords:

- Java
- Web Application
- Three-Tier Architecture
- JSP
- EJB
- RUP

1. Uvod

Spletne aplikacije postajajo v današnjih časih vedno bolj napredne in uporabnikom nudijo vedno več funkcionalnosti. Prav tako je delo z njimi lažje, če si izberemo primerne tehnologije za razvoj. Izdelavi in načrtovanju velikih projektov je težje slediti, zato smo primorani uporabljati eno izmed naštetih metodologij pri razvoju projekta. Izbrali smo metodologijo RUP, ki je enostavna in primerna za načrtovanje naše spletne aplikacije.

Zaradi današnjega načina življenja so spletne aplikacije hitra rešitev za človeka, ki je prezaseden in nima veliko časa, da na »običajen« način rezervira smučarski apartma. Uporabnik potrebuje samo internetno povezavo in že lahko brska in rezervira želeno lokacijo za zasluženi oddih.

Namen diplomskega dela je prikaz in izdelava spletne aplikacije, s katero uporabnik lahko na preprost način rezervira smučarski apartma v želenem terminu.

Spletni obiskovalec lahko preprosto rezervira apartma na kateremkoli razpisanem kraju, ki mu ga ta aplikacija ponuja, če je želeni termin takrat prost. Ko spletni obiskovalec potrdi želeno rezervacijo, dobi elektronsko pošto o potrditvi ali zavrnitvi rezervacije.

Prav tako dobi lastnik z elektronsko pošto obvestilo o morebitni rezervaciji enega ali več apartmajev. Nato preveri resničnost podatkov in potrdi ali zavrne rezervacijo apartmaja. Lastnik ima v aplikaciji natančen pregled nad vsemi smučarskimi kočami in apartmaji. To so podatki o datumu prihoda, stanju bivanja in datumu odhoda. Poleg tega ima možnosti urejanja podatkov o apartmajih in spreminjanja cene bivanja. Če ima lastnik smučarskih koč v prihodnosti namen prenavljanja ali celo investiranja v novo kočo, lahko vse spremembe objavi v aplikaciji. Ko je sprememba v aplikaciji potrjena, to vidi uporabnik pri sami rezervaciji apartmaja. Aplikacija je popolnoma avtomatizirana v smislu obveščanja o novih rezervacijah in potrditvah rezervacij preko elektronske pošte.

Cilji diplomskega dela so:

- prikaz razvoja spletne strani in kontrol s pomočjo ogrodja Struts,
- prikaz razvoja poslovne logike s pomočjo tehnologije strežniških Javanskih zrn (EJB),
- konkreten prikaz razvoja spletne aplikacije po metodologiji RUP.

2. Opis tehnologij in orodij za razvoj spletne aplikacije

V tem poglavju bomo opisali tehnologije in orodja za razvoj aplikacije, ki smo jih uporabljali za načrtovanje in razvoj spletne aplikacije.

Poglavji se razdeli na dve podpoglavji:

- razvojna orodja in
- opis tehnologij.

2.1 Razvojna orodja

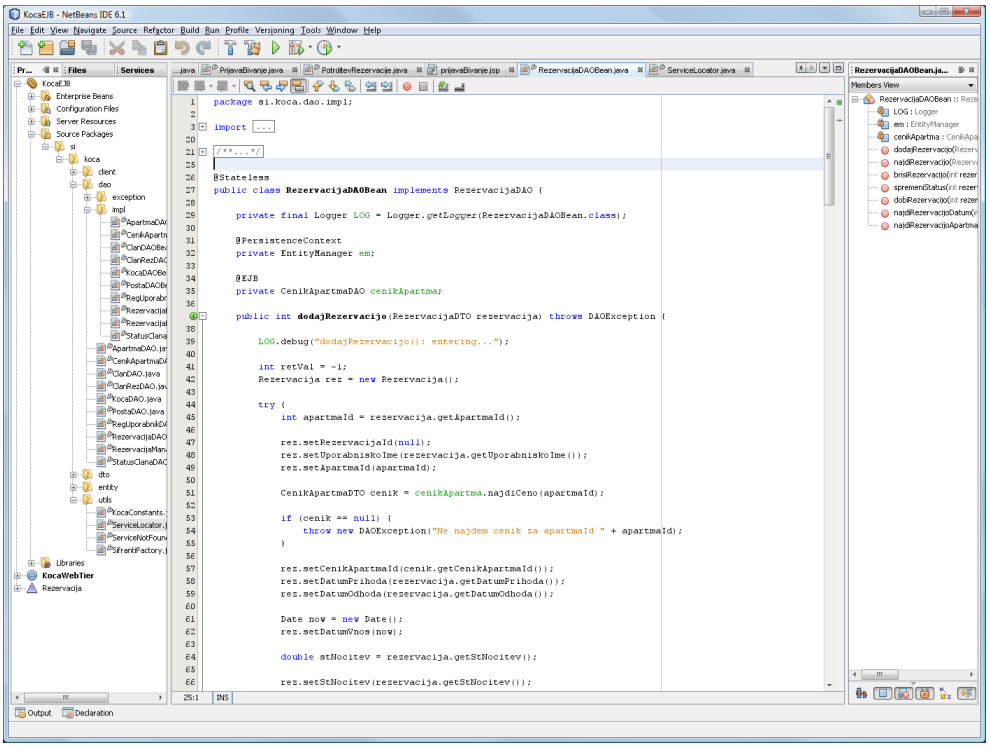
To so integrirana razvojna orodja, ki zagotavljajo celovite rešitve za razvoj programske opreme.

V tem poglavju bomo opisali:

- NetBeans IDE,
- Oracle JDeveloper in
- Visual Paradigm for UML.

2.1.1 NetBeans IDE

NetBeans IDE [8] je odprto-kodno integrirano razvojno okolje za razvoj Javanskih aplikacij (slika 1). Napisano je v celoti v programskem jeziku Java. Teče na mnogih platformah, vključno z Windows, Linux, Mac OS X in Solaris.



Slika 1: Razvojno orodje NetBeans IDE

NetBeans IDE omogoča uporabnikom široko paleto funkcij za izdelavo [9]:

- **Java namiznih aplikacij** – razvijanje konzolnih aplikacij in aplikacij z grafičnim uporabniškim vmesnikom,
- **Java EE in spletnih aplikacij** – celotno zbirko orodij za razvoj Java EE aplikacij, izdelava spletnih aplikacij z uporabo Ajax, JavaScript in CSS tehnologij,
- **vizualni mobilni razvoj** – ustvarjanje, preizkušanje in razhroščevanje GUI aplikacij, ki tečejo na mobilnih telefonih in dlančnikih.

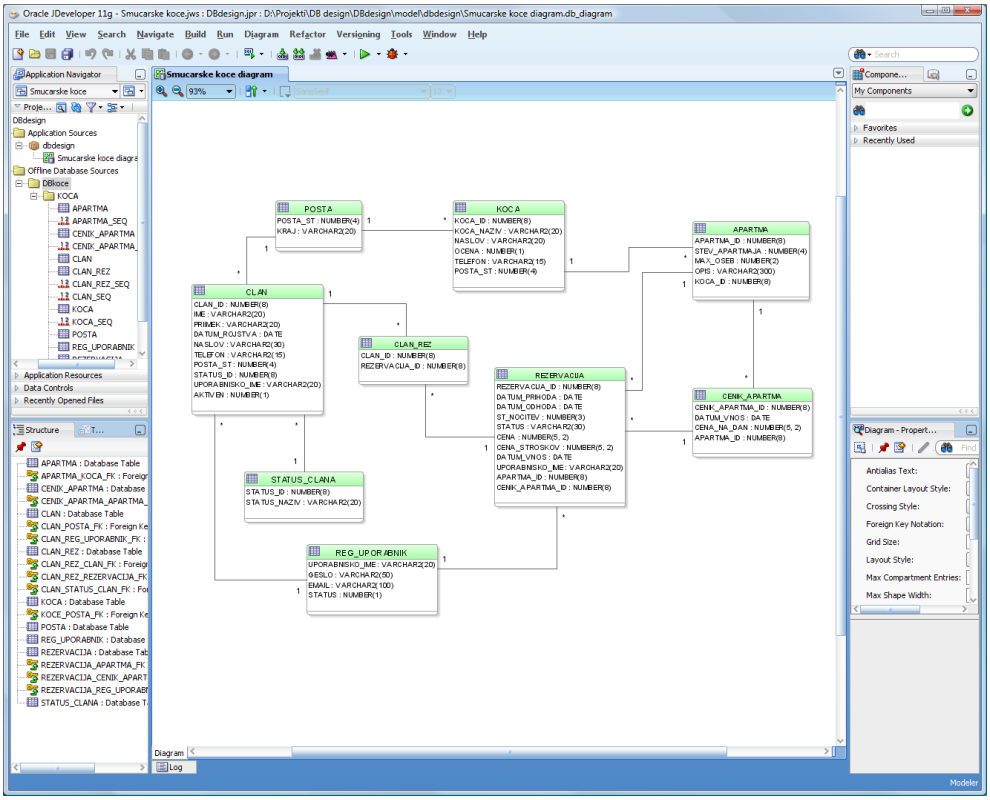
NetBeans IDE vsebuje različne elemente, ki jih potrebujemo za hiter in učinkovit razvoj aplikacij:

- prevajalnik kode,
- refaktoriranje kode,
- razhroščevalnik,
- kontrolo izvirnih datotek in verzij,
- testiranje enot,
- dopolnjevalec kode,
- vključitev različnih podatkovnih baz,
- določanje vira za aplikacijski strežnik, tako da lahko med razvojem poženemo aplikacijo neposredno iz delovnega okolja.

NetBeans IDE je zelo zmogljivo orodje ravno zato, ker zanj lahko dograjujemo množice dodatkov (ang. plugin) različnih razvijalcev, s katerimi si olajšamo delo.

2.1.2 Oracle JDeveloper

Oracle JDeveloper [11] je brezplačno orodje podjetja Oracle, ki je specializirano za razvoj in trženje programske opreme.



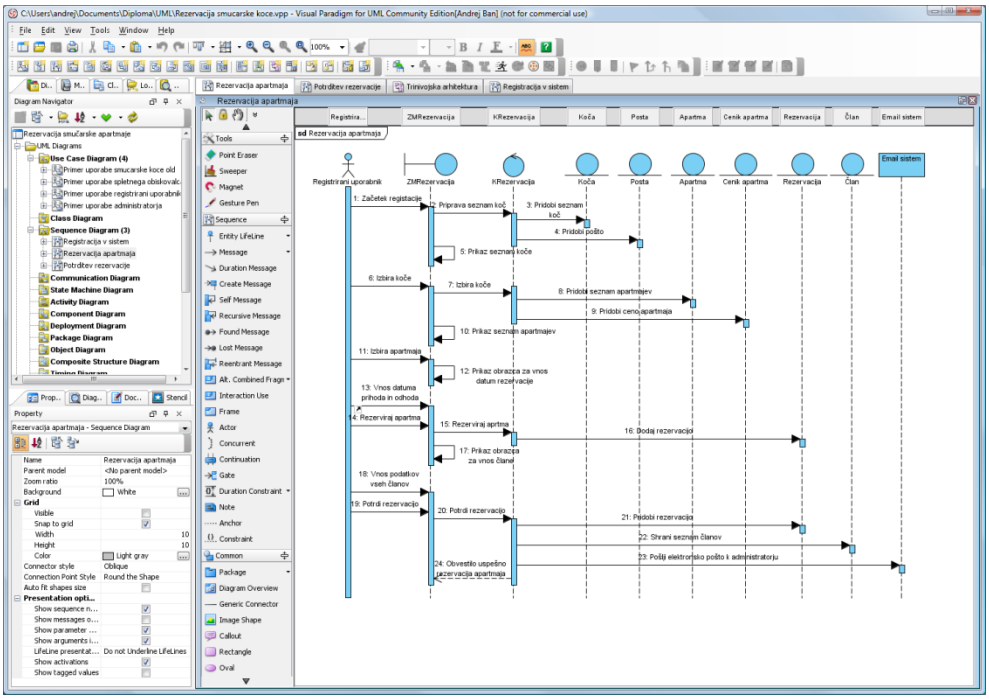
Slika 2: Razvojno orodje Oracle JDeveloper

Oracle JDeveloper je integrirano razvojno okolje (IDE) za izgradnjo aplikacij in spletnih storitev z uporabo najnovejših standardov za Java, XML in SQL. Oracle JDeveloper podpira razvoj v vseh fazah življenjskega cikla in obsega funkcije za modeliranje, kodiranje, razhroščevanje, testiranje profilov, nastavitev in namestitev aplikacije. Oracle JDeveloper omogoča tudi popoln razvoj in modeliranje okolja za izgradnjo baze podatkov.

To razvojno orodje smo uporabljali samo za načrtovanje in izdelavo podatkovnega modela naše aplikacije. Ker to orodje in podatkovna baza izhajata iz istega podjetja, je odlična kombinacija za povezovanje med njima, ki omogoča hiter razvoj podatkovne baze.

2.1.3 Visual Paradigm for UML

Visual Paradigm for UML [13] je programska oprema za vizualno UML modeliranje, ki deluje na različnih platformah. Poleg UML modeliranja podpira še modeliranje diagramov podatkovnih baz, vsebuje tudi generator izvorne kode za aplikacije Java in okolja .NET.



Slika 3: Orodje Visual Paradigm for UML

Orodje Visual Paradigm for UML podpira 13 diagramskih tehnik UML-ja:

- razredne diagrame,
- diagrame primerov uporabe,
- diagrame zaporedja,
- diagrame komunikacije,
- diagrame stanj,
- diagrame aktivnosti,
- komponentne diagrame,
- diagrame namestitve,
- diagrame paketov,
- objektne diagrame,
- diagrame sestavljenih struktur,
- časovne diagrame ter
- pregledni diagram interakcije.

Orodje Visual Paradigm je na voljo v več različicah, od osnovnih do profesionalnih. Obstaja tudi brezplačna (ang. Community Edition) za nekomercialno uporabo. Pri brezplačni različici ne moremo ustvariti izvorne kode in tabel podatkovne baze, vendar tudi ta ponuja precej funkcij in nam zadošča za načrtovanje in modeliranje diagramov.

2.2 Opis tehnologij

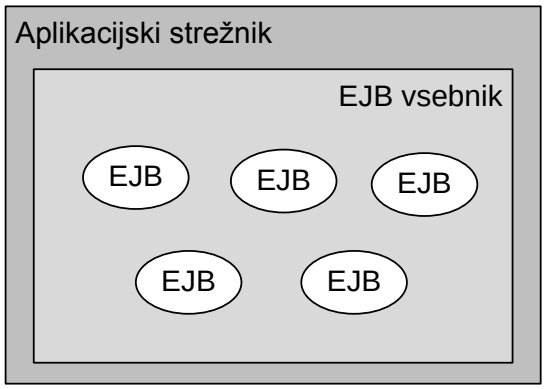
V tem poglavju bomo opisali tehnologije, ki smo jih uporabili za izdelavo spletne aplikacije.

2.2.1 Strežniška javanska zrna – EJB 3.0

Strežniška javanska zrna so strežniška komponenta tehnologije Java EE verzije 5 za razvoj poslovne aplikacije. Uporabljamo jo v večnivojskem okolju za rešitev poslovne logike in poslovnih podatkov.

EJB arhitekturo (slika 4) sestavlja:

- Java EE aplikacijski strežnik,
- izvajalno okolje – vsebnik za EJB komponente ter
- EJB komponente.



Slika 4: EJB arhitektura

EJB vsebnik je izvajalsko okolje za aplikacijske komponente in predstavlja enoten pogled na podrejene knjižnice (API) [1]. Takšna arhitektura povezuje aplikacijske komponente in storitve, ki jih predpisuje specifikacija, in proženje storitev napravi transparentno. EJB vsebnik prispeva zrna z različnimi storitvami, kot so: življenjski cikel upravljanja, varnost, upravljanje transakcij in še veliko več.

Ustvarjalci EJB zrn so razvili več verzij. Največ sprememb specifikacij je bil deležen ob prehodu iz verzij 2.1 v 3.0. Razvijalcu je treba pri razvijanju EJB 3.0 zrn postoriti manj stvari, vsebnik EJB zrn pa ponuja več storitev [3] . To je doseženo s pomočjo razvoja komponent v

obliki navadnih javanskih objektov POJO (ang. Plain Old Java Objects) in uporabe oznak (ang. Annotation). Na POJO temelječ razvoj ponuja več svobode pri razvoju in manj zahtev, ki jih mora razvijalec upoštevati. Pri uporabi oznake se zmanjša potreba po namestitvenih deskriptorjih.

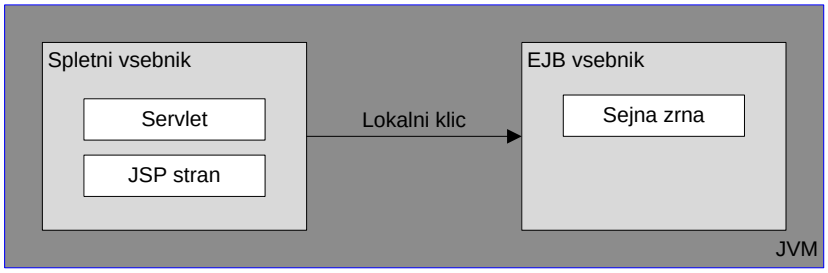
EJB zrno sestavlja:

- poslovni vmesnik in
- implementacijski razred.

Primer programske kode sejnega zrna po specifikaciji EJB 3.0:

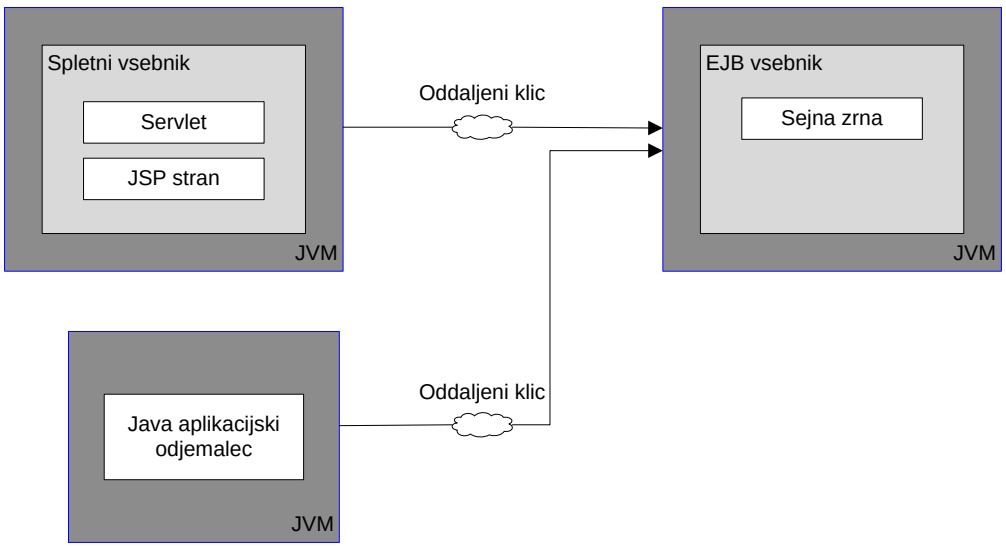
```
@Remote ali
@Local
public interface ApartmaDAO {
 public void dodajApartma(String naziv, String naslov…) throws Exception;
}

@Stateless
public class ApartmaDAOBean implements ApartmaDAO {
 public void dodajApartma(String naziv, String naslov…) throws Exception {
  // poslovna logika
 }
}
```

Poslovni vmesnik lahko obstaja v lokalni in oddaljeni različici (oznaki *@Local* in *@Remote*). Odjemalec pri komunikaciji z EJB zrni potrebuje vmesnike strežniških zrn in podatke za povezavo z JNDI strežnikom. JNDI je Java EE standardni vmesnik za lociranje uporabnikov, računalnikov, omrežij, objektov in storitev. JNDI v Java EE uporabljamo za pridobivanje referenc na oddaljene objekte (RMI, EJB), uporabniško definiranih transakcij in povezav do podatkovnih virov.
Pri lokalni različici lahko odjemalec komunicira z EJB zrnom le v primeru, če sta odjemalec in EJB vsebnik (slika 5) v istem izvajalnem okolju (JVM) [4].



Slika 5: Lokalni klic med odjemalcem in sejnim zrnom

Slika 6 pa prikazuje oddaljeno različico, kjer odjemalec oddaljeno kliče metodo sejnih zrn. Odjemalci in EJB vsebnik imata lahko različne verzije izvajalnega okolja.



Slika 6: Oddaljeni klici med odjemalci in sejnimi zrni

Lokalno izvajanje metod je veliko hitrejše kot oddaljeno, saj pri lokalnem neposredno kličemo metode brez uporabe protokola RMI-IIOP.

RMI-IIOP [12] je protokol, ki ga uporabljajo EJB komponente za oddaljeno komunikacijo. Predstavlja posebno različico RMI, ki je skladna s CORBA specifikacijo.

Strežniška javanska zrna delimo na:

- sejna zrna,
- sporočilna zrna,
- entitete in
- entitetna zrna (stvar preteklosti - v Java EE 5 so vključena le zaradi združljivosti).

Sejna zrna (ang. Session Beans)

Sejna zrna so namenjena modeliranju poslovnih procesov. Predstavljajo akcije oz. aktivnosti v sistemu (npr. avtorizacija uporabnika, rezerviranje apartmajev…).

Imamo dve vrsti sejnih zrn:

- sejna zrna brez stanja (ang. Stateless Session Beans) – ne ohranjajo sejnega stanja,
- sejna zrna s stanjem (ang. Statefull Session Beans) – ohranjajo stanje sejnih podatkov na nivoju ene uporabniške seje. Primer: začni nakupovanje, dodaj izdelke v košarico ter zaključi nakupovanje.

Tip sejnega zrna označimo v Javi razredu s pomočjo oznak `@Stateless` in `@Stateful`.

Sporočilna zrna (ang. Message Driven Beans)

Sporočilna zrna so namenjena modeliranju asinhrono proženih akcij oz. aktivnosti (npr. akcijsko znižanje cen, preklic kreditne kartice, objava ponudbe vrednostnih papirjev…). Sporočilna zrna lahko aktiviramo samo s poslanim sporočilom v čakalni vrsti.

Entitete (ang. Entity)

Entitete so Java razredi, namenjeni za delo s trajnimi objekti. Njihovo stanje se skladišči izven aplikacije, običajno v relacijski podatkovni bazi [4]. Za razliko od sejnega zrna entitete nimajo poslovne logike, kot je recimo shranjevanje, poizvedovanje, posodobitev in brisanje. Zato so entitete namenjene za delo s trajnimi objekti in so v sklopu specifikacije EJB pakirane v ločeni specifikaciji, imenovani Java Persistence API (JPA). JPA se uporablja znotraj in izven konteksta EJB vsebnika.

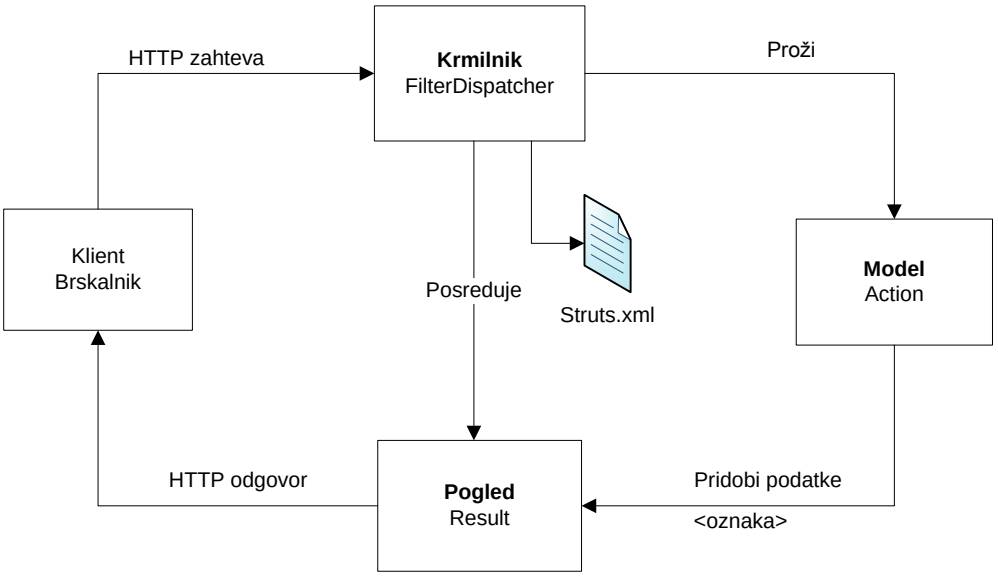
Java Persistence API [3] je orodje, ki je namenjeno delu s trajnimi objekti. Z uporabo JDBC tehnologije, objektno relacijskega preslikovanja, oznak in opcijskih konfiguracijskih datotek skrbi za enostavno in transparentno hranjenje objektov v relacijskih podatkovnih bazah. JPA je uporaben tako v sklopu platforme Java EE kot tudi SE in postaja standardna rešitev dostopa do relacijskih podatkovnih baz na visokem aplikacijskem nivoju. Eden od glavnih konceptov JPA je upravljalec persistence (ang. Persistence Manager). Z njegovo pomočjo lahko objekte umeščamo v trajen kontekst, jih iz njega začasno ali trajno umikamo, po njih poizvedujemo s pomočjo naprednega poizvedovalnega jezika in podobno.

2.2.2 Ogrodje Struts 2

Ogrodje Struts 2 [2] je zelo prožen in eleganten način za razvoj javanskih spletnih aplikacij za podjetja vseh velikosti in ni del platforme Java EE. Ogrodje Struts je bilo izdelano, da bi razvijalcem olajšalo razvoj kvalitetnih zahtevnih spletnih aplikacij, ki temeljijo na servletih, straneh JSP in lastnih oznakah. Gre za odprtokodno ogrodje, ki temelji na vzorcu **model-pogled-krmilnik** (ang. Model-View-Controller). **Model** predstavlja poslovno logiko, ki skrbi za hranjenje in obdelavo podatkov. **Pogled** predstavlja dostop do podatkov modela in jih ustrezno prikazuje. V primeru sprememb modela se pogled ustrezno osveži. Naloga **krmilnika** je, da se spremembe v uporabniškem vmesniku ustrezno prenesejo na model.

Struts 2 obsega tri glavne komponente (slika 7):

- `FilterDispatcher` – krmilnik,
- `Result` – pogled,
- `Action` – model.



Slika 7: Struts 2 obsega tri komponente: krmilnik, model in pogled

Tok operacij za Struts 2 je naslednji: Uporabnik pošlje zahtevek z URL-jem (http://www.mojastran.si/preveri.action). Krmilnik filtrira zahtevke, preveri URL in ga poišče v mapiranih akcijah, definiranih v datoteki `struts.xml`. Servlet najde instanco mapiranega akcijskega razreda (model) in pokliče metodo `execute()`. Metoda `execute()` izvede poslovno logiko za delo z podatkovnimi bazami, recimo shranjevanje ali pridobivanje podatkov iz baze. Ko model zaključi s poslovno logiko, preveri, kakšen rezultat vrne metoda, nato izbere pogled, ki je določen v datoteki `struts.xml`. Na koncu generira JSP stran na podlagi modela in rezultat se vrne k uporabniku.

Naloge razvijalca za pripravo ogrodja Struts 2 so:

- implementirati akcijski razred (razširiti osnovne `ActionSupport` razred) za vsako prejeto logično zahtevo in napisati v razredu metodo `execute()`,
- napisati konfiguracijsko datoteko za akcijo – `struts.xml`,
- posodobiti aplikacijsko datoteko `web.xml`, tako da v njej definiramo razred `FilterDispatcher`.

Glavne pridobitve Struts 2 so:

- zmanjšana kompleksnost – ni nam potrebno graditi lastnega MVC orodja,
- spodbuja dobre prakse (vzorci),
- enostavna uporaba,
- ponuja veliko funkcij,
- vključuje celo paleto tretjih aplikacij,
- razširljivost,
- odprtokodnost,

- dobra povezljivost z Java EE,
- skupna obdelava napak.

2.2.3 Aplikacijski strežnik JBoss

JBoss je najbolj razširjen odprto kodni aplikacijski strežnik na trgu. Aplikacijski strežnik JBoss [6] lahko namestimo na več operacijskih sistemov, kjer je prisotna JVM. JBoss je v celoti kompatibilen z Java EE, vključno z naprednimi storitvami, kot so: gruče (ang. Clustering), spletne storitve, strežniška Javanska zrna (EJB), sporočilna zrna (ang. Java Message Service), delo s transakcijami (ang. Java Transaction API) in drugo. JBoss ima vgrajen spletni strežnik Apache Tomcat, na katerem se izvajajo HTML strani, JSP strani in servleti.

2.2.4 Podatkovna baza Oracle DB

Podatkovna baza Oracle [10] je najpomembnejši produkt podjetja Oracle. Podatkovna baza Oracle je vodilni sistem za upravljanje z bazami podatkov, ki omogoča shranjevanje ogromne količine podatkov, dokumente, večpredstavnostne datoteke in XML. Na voljo je na različnih operacijskih sistemih in različno velikih platformah, od enoprocesorskih sistemov do mrež medsebojno povezanih strežnikov. Oracle bazo je mogoče enostavno vzpostaviti in upravljati, sposobna je izvajati različno zahtevna opravila in ponuja izredno dostopnost, učinkovitost, razširljivost, zanesljivost ter varnost.

3. Razvoj aplikacije za rezervacijo smučarskega apartmaja

V prejšnjem poglavju smo opisali orodja in tehnologije, s katerimi smo načrtovali in razvili našo aplikacijo. V tem poglavju bomo predstavili metodologijo razvoja aplikacij, ki smo jo uporabili pri razvoju naše aplikacije.

3.1 Razvoj aplikacije po metodologiji RUP

Metodologija Rational Unified Process je procesna platforma za razvoj informacijskih sistemov. Glavni namen uporabe metodologije RUP je uspešen razvoj programske opreme.

Metodologija RUP [7] je rezultat prizadevanja za bolj učinkovito objektno usmerjen razvoj programske opreme na ravni organizacije. Podobno kot druge metodologije ima natančno definirane metode, tehnike in proces razvoja ter določa modele.

RUP opisuje, kako učinkovito uporabiti šest najboljših izkušenj s področja razvoja programske opreme [5]:

- **iterativni razvoj** – temelji na izvajanju več iteracij oziroma ponovitev, kjer v vsaki iteraciji razvijemo del funkcionalnosti sistema,
- **obvladovanje zahtev** – tukaj gre za sledenje zahtev od začetka, preko sprememb do zaključka,
- **uporabo komponente arhitekture** – razbijanje razvoja na posamezne dele, kar zmanjša zapletenost sistema,
- **vizualno modeliranje** – prikaz modela sistema z uporabo jezika UML-ja,
- **preverjanje kakovosti** – gre za ocenjevanje kakovosti izdelka z vidika funkcionalnosti, zanesljivosti in zmogljivosti sistema,
- **nadzorovanje sprememb** – tukaj gre za redno spremljanje in poročanje. Pri nadzoru pomagajo struktura načrtov in vnaprej določena poročila.

RUP zajema štiri faze življenjskega cikla:

- **Začetna faza:** začetek projekta, zbiranje informacij, opredelitev okvirjev obravnavanega področja, študija izvedljivosti, definicija obsega projekta in utemeljitev poslovne pridobitve.
- **Zbiranje informacij:** analiza in načrtovanje obravnavanega področja, specifikacija značilnosti, načrtovanje arhitekture.
- **Konstrukcija:** konstrukcija zajema kodiranje in izdelavo izdelka. Konstrukcija lahko poteka v več iteracijah, če gre za velik projekt.
- **Prevzem:** priprava za prenos h končnemu uporabniku, namestitev pri uporabniku.

3.2 Faza 1 – začetna faza

Vsak razvoj se začne z začetno fazo. V začetni fazi zbiramo ključne elemente, ki so pomembni za sistem.

Cilji začetne faze so:

- vzpostavitev obsega programske opreme za projekt in vzpostavitev pogojev, ki vključujejo vizijo in kaj vse je v produktu zaželeno in kaj ne,
- določitev kritičnih primerov uporabe sistema, primarni scenarij operacij, ki bodo vodili h glavnemu načrtovanju,
- predstavitev ali demonstracija uporabe,
- ocenitev celotnih stroškov in urnikov za celoten projekt,
- ocenitev možnih tveganj in
- priprava podpornega okolja za projekt.

3.2.1 Primer uporabe

Primeri uporabe so ključni element RUP. Z diagramom primerov uporabe prikažemo, zakaj se sistem uporablja, kakšna je povezava med primeri uporabe in akterji. Akter lahko predstavlja osebo, napravo ali stvar izven sistema, ki je v komunikaciji s sistemom in s tem vpliva na njegovo delovanje. Primer uporabe je zaporedje akcij, ki jih izvede sistem in daje določenemu akterju nek rezultat [5].

Z jezikom UML smo zgradili diagram primerov uporabe. Akterji, ki nastopajo v naši aplikaciji, so:

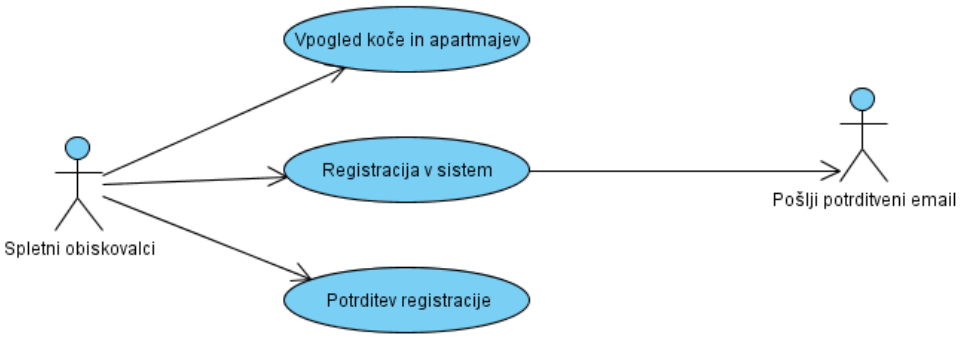
- spletni obiskovalci,
- registrirani uporabniki ter
- administrator.

3.2.1.1 Primeri uporabe za spletnega obiskovalca

Primeri uporabe za spletnega obiskovalca (slika 8) so naslednji:

- Prvi primer uporabe je **pregled koče in apartmajev,** kjer lahko spletni obiskovalci pregledajo seznam vseh smučarskih koč in vsako kočo posebej s seznamom vseh njenih apartmajev.
- Drugi primer uporabe je **registracija v sistem.** Če želi uporabnik rezervirati apartmaje, se mora najprej registrirati v sistem. Tukaj vnese podatke, kot so uporabniško ime, geslo in elektronski naslov. Če je bila registracija uspešno izvedena, dobi uporabnik aktivacijsko povezavo na svojem elektronskem naslovu.

- Tretji primer uporabe je **potrditev registracije**, kjer po prejetju elektronske pošte potrdi aktivacijsko povezavo. Če se je aktivacijska povezava uspešno izvedla, se lahko uporabnik že prijavi v spletno aplikacijo.

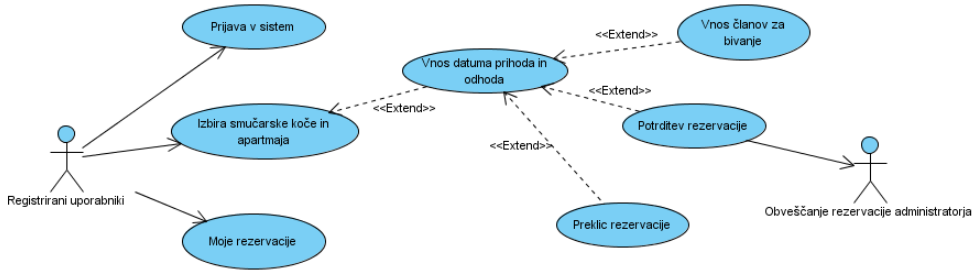


Slika 8: Primer uporabe za spletnega obiskovalca

3.2.1.2 Primeri uporabe za registriranega uporabnika

Primeri uporabe za registrirane uporabnike (slika 9) so naslednji:

- Prvi primer uporabe je **prijava v sistem,** kjer se uporabnik prijavi v sistem z uporabniškim imenom in geslom.
- Drugi primer uporabe je **izbira smučarske koče in apartmaja,** kjer uporabnik izbere kočo in apartma, da bi nato potrdil svojo rezervacijo.
- Tretji primer uporabe je **vnos datuma prihoda in odhoda,** kjer uporabnik vnaša datum prihoda in odhoda izbranega apartmaja. Če je termin prost, sistem doda novo rezervacijo v podatkovno bazo, v nasprotnem primeru pa sistem opozori, da je termin zaseden.
- Četrti primer uporabe je **vnos članov za bivanje,** kjer uporabnik vnaša seznam članov, ki bodo bivali v apartmaju.
- Peti primer uporabe je **potrditev rezervacije.** Tukaj uporabnik potrdi rezervacijo. Če je rezervacija uspešno zaključena, sistem pošlje elektronsko pošto administratorju o novi rezervaciji smučarskega apartmaja.
- Šesti primer uporabe je **preklic rezervacije.** Če si uporabnik premisli in ne želi rezervirati apartmaja, izvede akcijo preklic rezervacije. Sistem odstrani rezervacijo iz podatkovne baze.
- Sedmi primer uporabe je - **moje rezervacije.** Tukaj lahko vsak registrirani uporabnik pogleda stanje za seznam svojih rezervacij, ki jih je administrator potrdil.

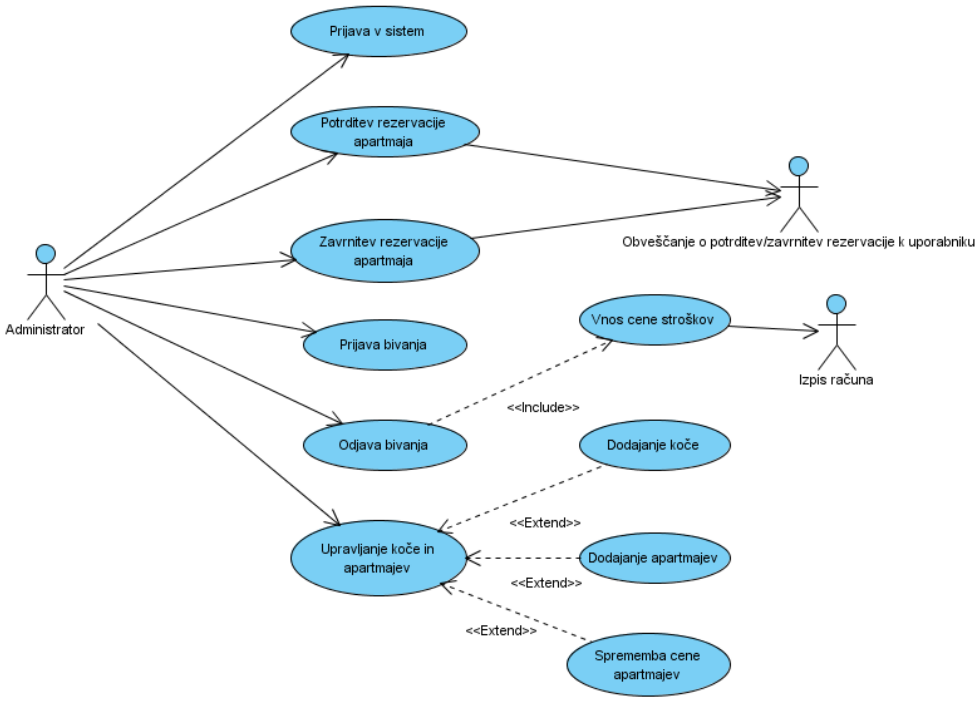


Slika 9: Primer uporabe za registriranega uporabnika

3.2.1.3 Primeri uporabe za administratorja

Primer uporabe za administratorja (slika 10) so:

- Prvi primer uporabe je **prijava v sistem,** kjer se uporabnik prijavi v sistem.
- Drugi primer uporabe je **potrditev rezervacije apartmaja.** Tukaj ima administrator seznam nepotrjenih različnih terminov rezervacij. Za vsak termin lahko podrobno pregleda pravilnost podatkov in nato potrdi rezervacijo apartmaja. Potem sistem avtomatsko pošlje uporabniku sporočilo o uspešni rezervaciji preko elektronske pošte.
- Tretji primer uporabe je **zavrnitev rezervacije apartmaja.** Če administrator ugotovi nepravilnosti ali manjkajoče podatke iz različnih razlogov, lahko zavrne rezervacijo apartmaja. Sistem pošlje sporočilo, da je bila rezervacija zavrnjena.
- Četrti primer uporabe je **prijava bivanja.** Ta pa se izvaja ob nastopu bivanja v apartmaju, administrator preveri, če se ujema datum začetka bivanja vseh članov, in potrdi prijavo bivanja kontaktne osebe. Nato lahko sledi predaja ključa.
- Peti primer uporabe je **odjava bivanja.** Ko nastopi čas odhoda iz apartmaja, administrator poišče termin, vnese dodatno še ceno stroška bivanja in zaključi bivanje v apartmaju. Sistem izračuna skupno ceno in ta se izpiše na zaslonu. Poleg tega ima administrator možnost izdelati račun v obliki PDF datoteke, ki jo lahko natisne.
- Šesti primer uporabe je **upravljanje koče in apartmaja.** V času vzpostavitve sistema, ko je še prazen, mora administrator vnesti seznam vseh smučarskih koč s potrebnimi podatki na različnih lokacijah, ki jih ima na voljo. Poleg vsake koče mora vnesti še seznam vseh apartmajev, ki jih vsebuje posamezna koča. V vsak apartma vnese tudi potrebne podatke, kot so opis apartmaja, velikost in ceno na noč/dan.
- Sedmi primer uporabe je **sprememba cene apartmaja.** Če administrator želi povečati ceno prenočišča apartmaja, to lahko stori tako, da izbere določeno kočo in apartma.



Slika 10: Primer uporabe za administratorja

3.3 Faza 2 – Analiza zahtev in načrtovanje

V procesu razvoja aplikacije sta analiza in načrtovanje najpomembnejša dejavnika pri uspešnosti izvedbe. Z analizo in načrtovanjem dosežemo učinkovito komuniciranje s strankami o želeni strukturi in obnašanju sistema, boljše razumevanje sistema in iskanje možnosti poenostavitve ter ponovne uporabe, vizualizacijo in nadzor nad arhitekturo sistema.

Za analizo in načrtovanje uporabljamo različne diagramske tehnike. Uporabljamo jih za opis specifikacij, načrtov in modelov.

3.3.1 Zahteve

V fazi zahteve smo se odločili, da mora aplikacija upoštevati naslednje zahteve:

- spletna aplikacija mora delovati v spletnem brskalniku Mozilla Firefox na kateremkoli operacijskem sistemu,
- spletna aplikacija mora podpirati delovanje več uporabnikov,
- spletna aplikacija mora biti varna pred morebitnimi zlorabami,
- spletna aplikacija naj bo čimbolj enostavna za uporabo (nekaj klikov do želenih rezultatov),
- spletna aplikacija naj bo narejena tako, da se lahko nadgrajuje ob spremembah poslovanja.

3.3.2 Načrtovanje sistema

Za načrtovanje sistema smo uporabili dva modela:

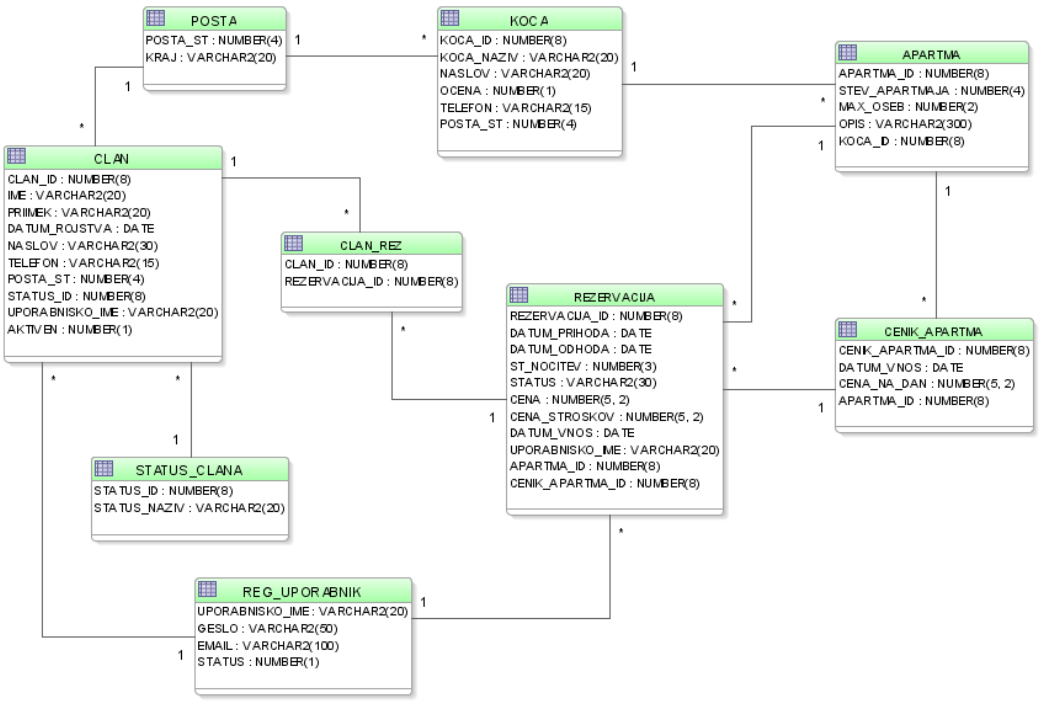
- podatkovni model in
- procesni model.

3.3.2.1 Podatkovni model

Glavni del diplomske naloge zajema podatkovni model sistema.
Podatkovni model je prikazan z:

- entitetami, ki vsebujejo atribute, in
- povezavami, ki prikazujejo medsebojno razmerje med entitetami.

Slika 11 prikazuje celotni podatkovni model za spletno aplikacijo rezervacije smučarskega apartmaja. Vsaka entiteta ima enolični primarni ključ.



Slika 11: Celotni podatkovni model

Podatkovni model (slika 11) vsebuje devet entitet, ki predstavljajo:

- *POSTA*: šifrant pošte s poštnimi številkami in krajem,
- *STATUS_CLANA*: šifrant status člana, ki določa naziv statusa člana (odrasli, študenti, otroci, dojenčki),
- *KOCA*: podatki o kočah,
- *APARTMA*: osnovni podatki o apartmaju, njegovi sestavi,
- *CENIK_APARTMA*: podatki o ceni prenočišča vsakega apartmaja,
- *REG_UPORABNIK*: podatki o registriranih uporabnikih, ki se registrirajo preko spletne aplikacije,
- *CLAN*: podatki o članih, ki bivajo v apartmaju,
- *REZERVACIJA*: to je glavna entiteta, kjer vodimo podatke o rezervaciji apartmaja,
- *CLAN_REZ*: to je vmesna entiteta, ki je povezana z entitetami *CLAN* in *REZERVACIJA*. Vsak član je lahko udeležen v več rezervacijah apartmaja, vsaka rezervacija pa ima tipično lahko več članov.

3.3.2.2 Procesni model

V tem poglavju bomo predstavili načrtovanje spletne aplikacije za rezervacije smučarskega apartmaja s pomočjo diagrama zaporedja. Diagram zaporedja opisuje dinamično obnašanje med akterji in sistemom ter med objekti sistema. Ti diagrami eksplicitno prikazujejo zaporedje sporočil in so primerni za prikaz celotnega toka dogodkov ter za specifikacijo dogodkov, ki potekajo v realnem času.

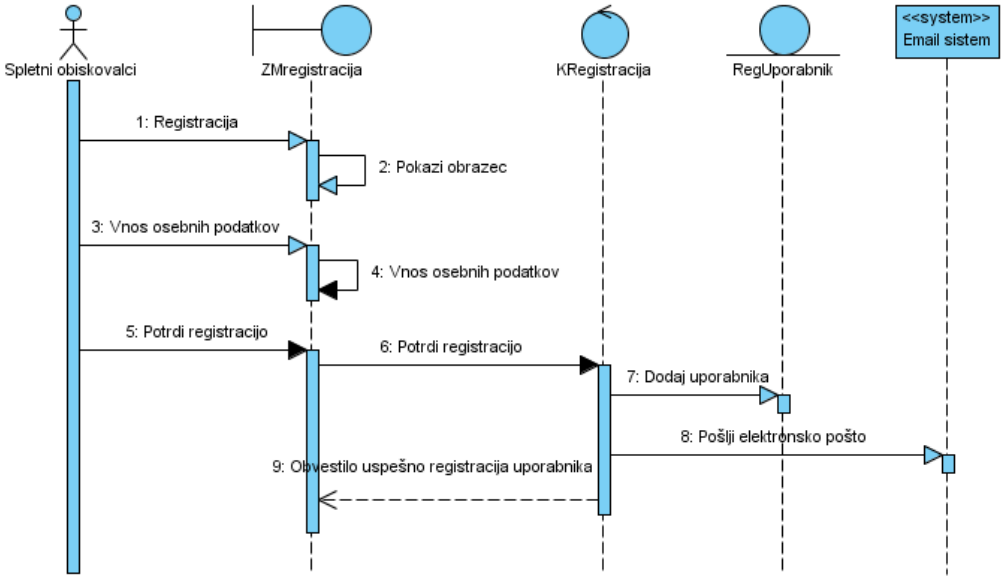
Zaradi obsežnosti aplikacije bomo v nadaljevanju opisali le tiste tehnike diagrama zaporedja, ki so posebej pomembne za našo spletno aplikacijo.

Ti diagrami zaporedja so:

- diagram zaporedja za registracijo v sistem,
- diagram zaporedja za rezervacijo apartmaja ter
- diagram zaporedja za potrditev rezervacije.

Diagram zaporedja za registracijo v sistem

Slika 12 prikazuje osnovni tok pri registraciji novega uporabnika v sistem.



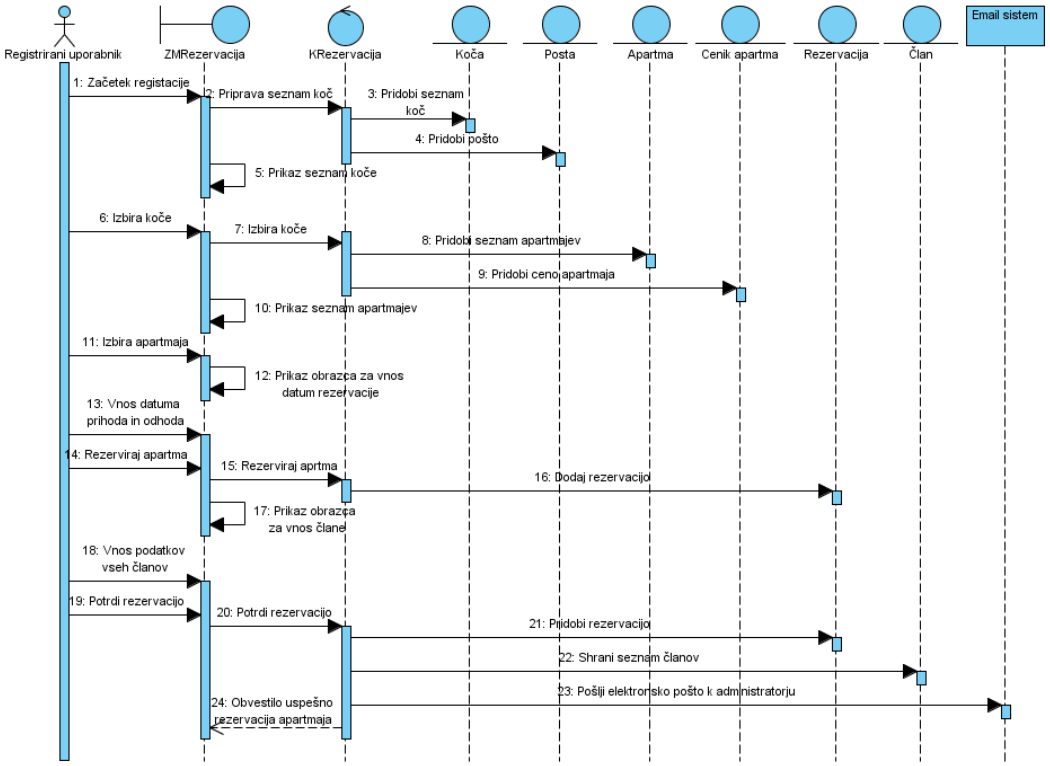
Slika 12: Diagram zaporedja za registracijo v sistem

Alternativni tokovi so naslednji:

- v primeru manjkajočih izpolnjenih podatkov nas sistem opozori ter
- v primeru nedostopne podatkovne baze prav tako.

Diagram zaporedja za rezervacijo apartmaja

Slika 13 prikazuje osnovni tok, kjer registrirani uporabnik rezervira apartma.



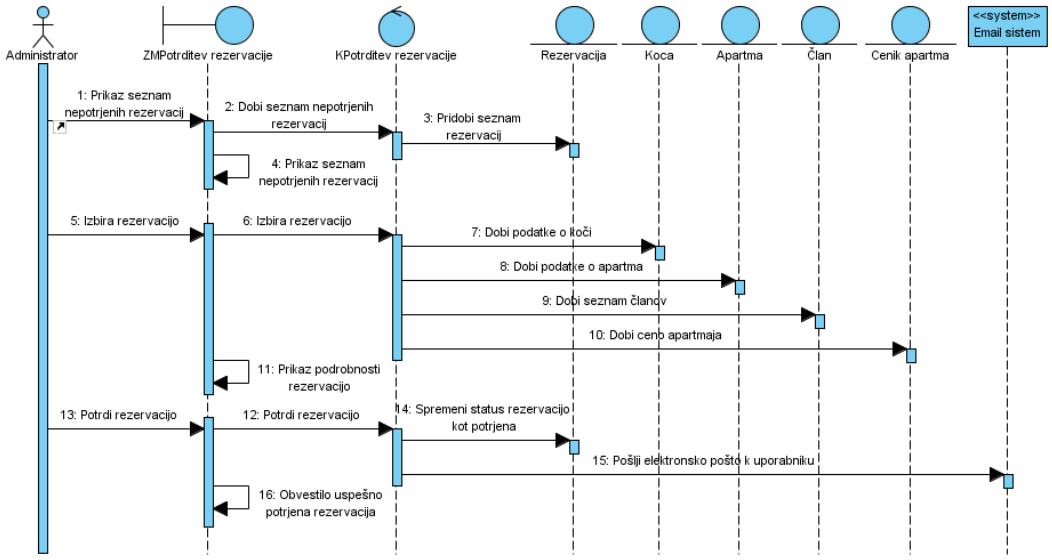
Slika 13: Diagram zaporedja za rezervacijo apartmaja

Alternativni tokovi za rezervacijo apartmaja, ob katerih nas sistem opozori, so:

- v primeru neizpolnjene vnosne maske datum prihoda in odhoda,
- če je datum prihoda in odhoda že zaseden,
- če je vnos datuma prihoda starejši od današnjega datuma,
- če je datum prihoda večji ali enak od datuma odhoda,
- če ne vnesemo nobenega člana,
- če je seznam članov večji od kapacitet rezerviranega apartmaja ter
- v primeru sistemske napake pri dostopu do podatkovnih baz.

Diagram zaporedja za potrditev rezervacije

Slika 14 prikazuje osnovni tok, kjer administrator potrdi rezervacijo apartmaja.



Slika 14: Diagram zaporedja za potrditev rezervacije

Alternativni tokovi za potrditev rezervacije so:

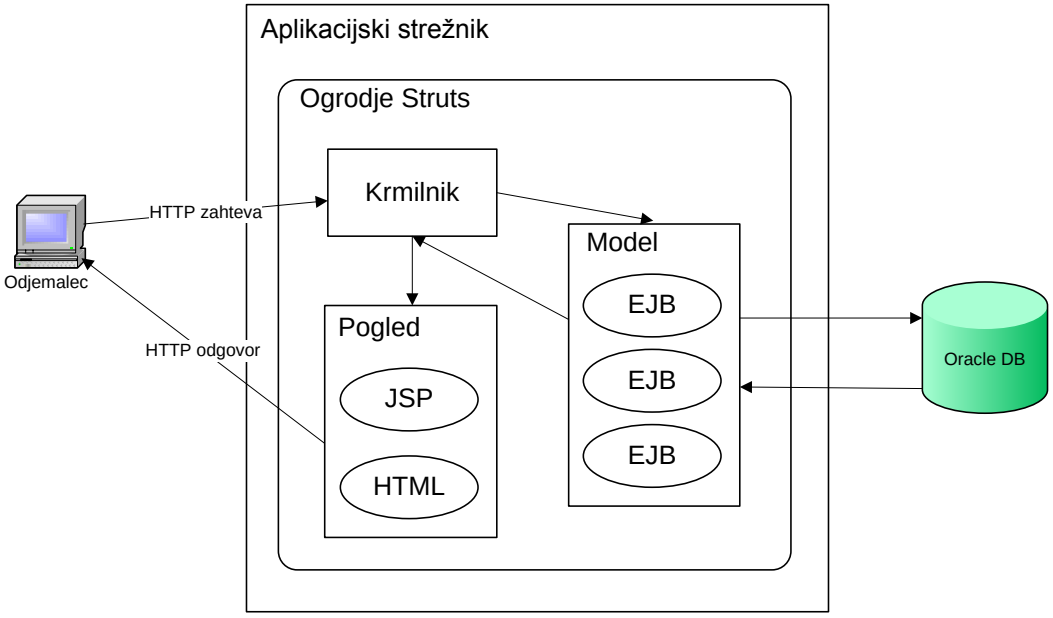
- če administrator zavrne rezervacijo, sistem pošlje elektronsko pošto k uporabniku,
- v primeru sistemske napake pri dostopu do podatkovnih baz nas sistem opozori.

3.4 Faza 3 – konstrukcija

Po opravljeni analizi in načrtovanju sistema je sledila izdelava aplikacije. Po premisleku smo se odločili, da bomo naredili aplikacijo v trinivojski arhitekturi (slika 15). Trinivojsko arhitekturo sestavljajo: podatkovna baza, aplikacijski strežnik in različni klienti.

Trinivojska arhitektura je katerikoli sistem, ki uveljavlja splošno ločitev naslednjih treh delov:

1. predstavitveni nivo ali nivo uporabniškega vmesnika,
2. nivo poslovne logike ali srednji nivo ter
3. podatkovni nivo.



Slika 15: Trinivojska ahritektura naše spletne aplikacije

3.4.1 Podatkovni nivo

Aplikacijski strežnik JBoss podpira vse glavne ponudnike podatkovnih baz (MS SQL, Oracle, PostgreSQL, MySQL, DB2). S tem je na voljo popolna svoboda pri izbiri poljubne podatkovne baze in celo pri eventualnih prihodnjih menjavah podatkovnih baz.

Za našo aplikacijo smo se odločili za podatkovno bazo Oracle. V razvojnem orodju Oracle JDeveloper smo načrtovali podatkovni model za našo aplikacijo. Orodje Oracle JDeveloper omogoča, da s čarovnikom, na podlagi načrtovanega podatkovnega modela, avtomatično generiramo fizični model podatkovne baze (tabele in atributi). S tem je podatkovna baza pripravljena za vnos podatkov.

3.4.2 Nivo poslovne logike

Nivo poslovne logike predstavlja glavno jedro naše aplikacije. Uporabili smo tehnologijo spletnega vmesnika JSP, ogrodje Struts2 in strežniških Javanskih zrn (EJB).

Najprej smo pripravili sloj za dostop do podatkovne baze z uporabo ogrodja Java Persistence API. Java Persistence API je ogrodje, namenjeno delu s trajnimi objekti.

V naši podatkovni bazi imamo devet tabel, torej moramo pripraviti devet entitet - javanskih razredov v obliki POJO. Te entitete so naslednje: *`Apartma`*, *`CenikApartma`*, *`Clan`*, *`ClanRez`*, *`Koca`*, *`Posta`*, *`RegUporabnik`*, *`Rezervacija`*, *`StatusClana`*.

Poglejmo si preprost primer – entiteto *Apartma*:

```
@Entity
@Table(name = "APARTMA")
@SequenceGenerator(name="apartmaSeq", sequenceName="APARTMA_SEQ")
public class Apartma implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy=GenerationType.SEQUENCE, generator="apartmaSeq")
    @Column(name = "APARTMA_ID", nullable = false)
    private Integer apartmaId;

    @Column(name = "STEV_APARTMAJA", nullable = false)
    private Integer stevApartmaja;

    @Column(name = "MAX_OSEB", nullable = false)
    private Integer maxOseb;

    @Column(name = "OPIS")
    private String opis;

    @Column(name = "KOCA_ID", nullable=false)
    private Integer kocaId;

    public Apartma() {
    }
    public Integer getApartmaId() {
        return apartmaId;
    }
    public void setApartmaId(Integer apartmaId) {
        this.apartmaId = apartmaId;
    }
    public Integer getStevApartmaja() {
        return stevApartmaja;
    }
    public void setStevApartmaja(Integer stevApartmaja) {
        this.stevApartmaja = stevApartmaja;
    }
    public Integer getMaxOseb() {
        return maxOseb;
    }
    public void setMaxOseb(Integer maxOseb) {
        this.maxOseb = maxOseb;
    }
    public String getOpis() {
        return opis;
    }
    public void setOpis(String opis) {
        this.opis = opis;
    }
    public Integer getKocaId() {
        return kocaId;
    }
    public void setKocaId(Integer kocaId) {
        this.kocaId = kocaId;
    }
}
```

Da navedeni javanski razred zares postane entiteta, ga moramo opremiti najmanj z oznako `@Entity` in enega od atributov označiti kot primarni ključ z oznako `@Id`. Podrobnejše informacije (kot so, kje naj se podatki hranijo, omejitve, imena in tipi stolpcev, načini povezovanja objektov, itd) določimo z množico oznak, kot so: `@Table`, `@Column`, `@SequenceGenerator`, `@GeneratedValue`, `@OneToMany`, `@ManyToOne` itd.

Uporaba entitet v trajnem kontekstu je nadvse preprosta in zahteva naslednje kode:

- pridobitev upravljalca presistence znotraj Java EE aplikacijskega strežnika:

```
@PersistenceContext
private EntityManager manger;
```

- umestitev v trajni kontekst:

```
Apartma a = new Apartma();
a.setApartmaId(null);
a.setStevApartmaja(1);
a.setMaxOseb(4);
a.setOpis("opis");
a.setKocaId(1);
manager.persist(a);
```

- umik iz trajnega konteksta:

```
manager.remove(a);
```

- poizvedovanje s pomočjo poizvedovalnega jezika JPA QL:

```
Query q = manager.createQuery("select a from Apartma a");
Collection<ApartmaDTO> listApartma = q.getResultList();
```

Nato je sledila implementacija strežniških javanskih zrn. V naši aplikaciji smo razvili devet sejnih zrn brez stanja in eno sejno zrno s stanjem. Sejna zrna smo uporabljali kot poslovno logiko za delo z navedenim ogrodjem JPA.

Osnovni razredi za sejno zrno brez stanja so (`si.koca.dao.impl` predstavlja ime paketa):

- `si.koca.dao.impl.ApartmaDAOBean` – vstopna točka v poslovno logiko za delo z apartmajem. Operacije so naslednje: dodajanje apartmaja, sprememba apartmaja in pridobivanje seznama apartmajev,
- `si.koca.dao.impl.CenikApartmaDAOBean` – vstopna točka v poslovno logiko za delo s cenikom apartmaja,
- `si.koca.dao.impl.ClanDAOBean` – poslovna logika za člane, ki bivajo v apartmaju,
- `si.koca.dao.impl.KocaDAOBean` – poslovna logika za upravljanje koče,
- `si.koca.dao.impl.PostaDAOBean` – poslovna logika za pridobitev seznama šifra pošt,

- `si.koca.dao.impl.RegUporabnikDAOBean` – poslovna logika za upravljanje registriranih uporabnikov v spletni aplikaciji,
- `si.koca.dao.impl.RezervacijaDAOBean` – poslovna logika za rezervacijo apartmajev,
- `si.koca.dao.impl.ClanRezDAOBean` – poslovna logika za pridobitev seznama članov iz določene rezervacije,
- `si.koca.dao.impl.StatusClanaDAOBean` – pridobitev seznama šifrant statusa članov.

Sejno zrno s stanjem:

- `si.koca.dao.impl.RezervacijaManagerDAOBean` – implementacija poslovnega razreda za upravljanje rezervacije. Uporabnik rezervira želeni apartma, dodaja člane za bivanje in potrjevanje/preklic rezervacije.

Implementirali smo Java razred `ServiceLocator`, ki služi za dostop do vseh sejnih zrn. Ta razred uporabljamo v naši aplikaciji pri krmilniku Struts za klic sejnega zrna. V naši aplikaciji gostuje ogrodje Struts in EJB skupaj v aplikacijskem strežniku JBoss. Razred `ServiceLocator` nam pride prav pri konfiguraciji, če bi kasneje želeli ogrodje Struts ločiti od EJB vsebnika na ločeni aplikacijski strežnik na drugem računalniku. Odjemalec dostopa do sejnega zrna s pomočjo klica oddaljenih metod (RMI-IIOP).

Primer programske kode za pridobivanje reference objekta in izvršitev metod:

```
InitialContext ctx = new InitialContext();
ApartmaDAO apartma = (ApartmaDAO)ctx.lookup("ApartmaDAOBean/remote");

apartma.dodajApartma("Apartma 11", "Trebež 3");
```

Ker imamo v naši aplikaciji ogrodje Struts in EJB na enem aplikacijskem strežniku JBoss, se klic sejnega zrna izvede lokalno, torej ni treba navesti parametrov za vir aplikacijskega strežnika v konstrukciji `InitialContext()`.

Nato je sledila izgradnja spletne aplikacije z ogrodjem Struts 2. To smo opravili v treh korakih:

- **izdelava pogled** – implementacija JSP datoteke,
- **izdelava razred Action** – predstavlja model za poslovno logiko aplikacije,
- **preslikava med akcijo in pogledom** – priprava konfiguracijske datoteke `struts.xml`, v kateri so zapisani podatki o povezavah med zahtevami in akcijami aplikacije, navigaciji med stranmi ipd.

V naši aplikaciji smo v datoteki `struts.xml` definirali tri pakete z oznakami `<package>`: za gosta, registriranega uporabnika in za administratorja.

```
<struts>
   <package name="gost" extends="struts-default">
   …
   </package>
   <package name="uporabnik" namespace="/user" extends="struts-default">
   …
   </package>
   <package name="admin" namespace="/admin" extends="struts-default">
   …
   </package>
</strtuts>
```

Oznaka *package* se uporablja za skupinsko konfiguriranje akcij, ki si delijo skupne lastnosti. V paketu *gost* prožijo akcije katerikoli spletni obiskovalci na internetu, paket *uporabnik* pa lahko sproži akcijo le v primeru, da so se že vnaprej prijavili v spletno aplikacijo, enako je tudi pri paketu *admin*. Preverjanje uporabnika izvedemo tako, da vključimo svoj prestreznik zahtev (ang. Interceptor). To storimo tako, da v paketu uporabnika in administratorja dodamo:

```
<interceptors>
     <interceptor name="avtorizacija"
         class="si.koca.interceptor.AvtorizacijaInterceptor"/>
     <interceptor-stack name="adminStack">
         <interceptor-ref name="avtorizacija"/>
         <interceptor-ref name="defaultStack"/>
     </interceptor-stack>
 </interceptors>

 <default-interceptor-ref name="adminStack"/>
```

Oznaka *defaultStack* je del ogrodja Struts, ki omogoča pridobivanje podatkov iz parametrov spletnega naslova, preverjanje pravilnosti podatkov, rokovanje izjem, nalaganje datotek itd.

Oznaka *avtorizacija* je naš prestreznik zahtev, ki najprej pokliče razred *AvtorizacijaInterceptor*, preden sproži določeno zahtevo za proženje akcij.

Spodnji del programske kode je primer preverjanja, če je uporabnik prijavljen v spletni aplikaciji.

```
public class AvtorizacijaInterceptor implements Interceptor {

    private static final Logger LOG =
             Logger.getLogger(AvtorizacijaInterceptor.class);

    public String intercept(ActionInvocation actionInvocation) throws Exception {

        LOG.debug("intercept(): entering...");

        ActionContext ctx = actionInvocation.getInvocationContext();
        Map session = ctx.getSession();

        String action = null;
```

```
        Uporabnik uporabnik = (Uporabnik)
               session.get(KocaConstants.UPORABNIK_SEJA);

        if (uporabnik != null) {

            if (actionInvocation.getProxy().getNamespace().equals("/admin")) {
                if (!uporabnik.isAdmin()) {
                    action = ActionSupport.LOGIN;
                }
            }
            if (action == null) {
                Action act = (Action)actionInvocation.getAction();
                if (act instanceof UporabnikAware) {
                   ((UporabnikAware) act).setUporabnik(uporabnik);
                }
                action = actionInvocation.invoke();
            }
        } else {
            ActionProxy proxy = actionInvocation.getProxy();
            if (proxy.getActionName().equals("rezervacija")) {

                proxy.setMethod("prijava");
                action = actionInvocation.invoke();
            } else {
                action = ActionSupport.LOGIN;
            }
        }

        LOG.debug("intercept(): leaving...");

        return action;
    }
}
```

Pri uspešni prijavi v spletni aplikaciji smo shranili objekt *Uporabnik* (vsebuje podatke o uporabniku) v podatkovno strukturo *Session*. Iz vmesnika *SessionAware* pridobimo sejo, ki hrani podatke nekega uporabnika med obdobjem, ko je prijavljen v spletni aplikaciji oziroma, dokler se uporabnik ne odjavi, zapusti spletni brskalnik ali pa je potekel čas seje (ang. Session timeout). V zgornji kodi je razvidno, da najprej pogledamo, če v objektu *Session* vsebuje razred *Uporabnik*. Če vsebuje, se sproži metoda *actionInvocation.invoke()* izbranega razreda (definirano v *struts.xml*), sicer pa se preusmeri v spletno stran »Prijava v sistem« *(ActionSupport.LOGIN)*.

Na nekatere spletne strani JSP imajo dostop samo registrirani in prijavljeni uporabniki. Uporabnik, ki bi poznal zgradbo spletne aplikacije in naslove posameznih strani, bi se lahko izognil prijavi, če bi šel direktno na ustrezni naslov. Da to preprečimo, v vsaki JSP strani najprej preverimo, če je uporabnik prijavljen. To dosežemo tako, da vstavimo spodnji del programske kode:

```
<s:if test="#session['uporabnik_seja'] == null">
    <% response.sendRedirect(request.getContextPath() + "/osnovna.action");%>
</s:if>
```

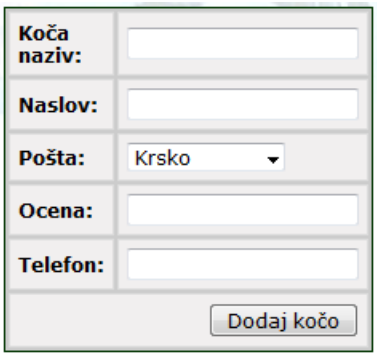
Pri prijavi se v spremenljivko *uporabnik_seja* v objektu *Session* zapiše šifra uporabnika. Preveriti moramo torej, če je ta spremenljivka definirana. Če ugotovimo, da ni, odjemalca preusmerimo na začetno stran z relativnim naslovom *osnovna.action*.

Pri razvoju modela smo implementirali 14 akcijskih razredov. Tu gre predvsem za procesiranje vnosa podatkov preko spletnega obrazca, preverjanje pravilnosti vnesenih podatkov in klic sejnih zrn za pridobivanje podatkov iz podatkovne baze.

- *si.koca.action.Registracija* – poslovna logika za registracijo v spletno aplikacijo. Po registraciji dobi uporabnik elektronsko pošto, ki vsebuje aktivacijsko spletno povezavo,
- *si.koca.action.PotrditevRegistracije* – poslovna logika za potrditev aktivacijske povezave, ki jo je uporabnik dobil preko elektronske pošte,
- *si.koca.action.SeznamKoce* – poslovna logika za pridobivanje seznama koč,
- *si.koca.action.SeznamApartma* – poslovna logika za pridobivanje seznama apartmajev izbrane koče,
- *si.koca.action.Prijava* – poslovna logika za prijavo v spletno aplikacijo. Po prijavi sistem obdrži sejo za identifikacijo uporabnika tako dolgo, dokler se ne odjavi iz sistema,
- *si.koca.action.Rezervacija* – poslovna logika za izdelavo mesečnega koledarja, ki vsebuje proste in zasedene dneve, vsebuje pa tudi proces za potrditev rezervacije apartmajev,
- *si.koca.action.Clani* – poslovna logika za dodajanje in odstranjevanje članov, ki bodo bivali v rezerviranem apartmaju, zaključitev rezervacije apartmaja ter preklic rezervacije apartmaja,
- *si.koca.action.MojeRezervacije* – poslovna logika, kjer uporabnik pridobi seznam vseh potrjenih rezervacij apartmajev,
- *si.koca.action.PotrditevRezervacije* – poslovna logika za prikaz seznama tistih uporabnikov, ki so v kratkem rezervirali apartmaje, potrditev rezervacije apartmaja oziroma zavrnitev rezervacije apartmaja,
- *si.koca.action.PrijavaBivanja* – poslovna logika za pridobivanje seznama vseh potrjenih uporabnikov za začetek bivanja v apartmaju, prikaz podrobnosti vsakega uporabnika in potrditev začetka bivanja v apartmaju,
- *si.koca.action.OdjavaBivanje* – poslovna logika za prikaz seznama uporabnikov, ki bivajo v apartmaju, prikaz podrobnosti o bivanju vsakega uporabnika, vnos cene za stroške bivanja in zaključek bivanja v apartmaju,
- *si.koca.action.Racun* – poslovna logika za izdelavo računa v datoteki PDF,
- *si.koca.action.Koce* – poslovna logika za prikaz spletnega obrazca za vnos koče, proces vnosa nove koče, sprememba podatkov obstoječih koč, prikaz spletnega obrazca za vnos apartmaja, proces vnosa novega apartmaja, sprememba podatkov obstoječega apartmaja, prikaz spletnega obrazca za vnos cene apartmaja in proces vnosa nove cene apartmaja,

- *si.koca.action.Odjava* – poslovna logika za odjavo iz spletne aplikacije. Po izvršitvi odjave se objekt *Session* uniči.

Za boljše razumevanje bomo spodaj prikazali celotni postopek gradnje Struts 2 za primer dodajanja nove koče.



Slika 16: Spletni obrazec za vnos nove koče

Najprej pripravimo JSP stran za dodajanje nove koče:

```
…
<%@ taglib prefix="s" uri="/struts-tags" %>

<html>
  <head>
    <title>Dodajanje koče</title>
  </head>

  <body>
    <s:form name="vnosKoce" action="vnosKoce" method="POST">
      <s:textfield label="Koča naziv" name="kocaNaziv"/>
      <s:textfield label="Naslov" name="naslov"/>
      <s:autocompleter label="Pošta" list="posta" listValue="kraj"
          listKey="postaSt" name="postaId"/>
      <s:textfield label="Ocena" name="ocena"/>
      <s:textfield label="Telefon" name="telefon"/>
      <s:submit value="Dodaj kočo"/>
    </s:form>
  …
  </body>
</html>
```

Nato konfiguriramo datoteko *struts.xml* za mapiranje posameznih akcij, ki izgleda takole:

```
<struts>
  <global-results>
    <result name="error">/napaka.jsp</result>
  </global-results>

  <global-exception-mappings>
    <exception-mapping exception="java.lang.Exception" result="error"/>
  </global-exception-mappings>
```

```
  <package name="gost" extends="struts-default">

    <action name="vnosKoce" class="si.koca.action.Koce">
      <result name="success">/kocaDodana.jsp</result>
      <result name="input">/vnosKoce.jsp</result>
      <result name="error">/napaka.jsp</result>
    </action>

    <action name="seznamKoce" … >
      …
    </action>

  </package>
</struts>
```

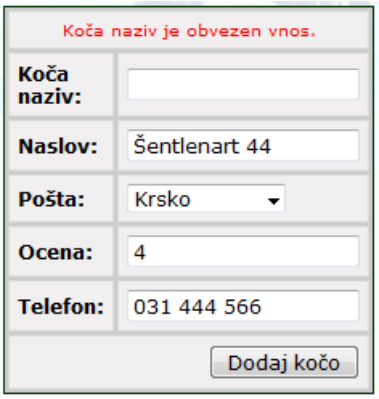
Pri sliki 16 se ob pritisku gumba **Dodaj kočo** sproži akcija s povezavo http://localhost/rezervacija/vnosKoce.action, kar je vidno iz tega obrazca (*<s:form name="vnosKoce" action="vnosKoce" method="POST">*), in poišče v akcijah, ki so definirane v datoteki *struts.xml*. Krmilnik najde instanco mapiranega akcijskega razreda (*<action name="vnosKoce" class="si.koca.action.Koce">*) in izvrši metodo *execute()*.

```
package si.koca.action;

import com.opensymphony.xwork2.ActionSupport;
…

public class Koce extends ActionSupport {

    private static final Logger LOG = Logger.getLogger(VnosKoce.class);

    private Integer kocaId;
    private String kocaNaziv;
    private String naslov;
    private Integer postaId;
    private Integer ocena;
    private String telefon;
    private Collection<PostaTO> posta;

    public String execute() throws Exception {

        LOG.debug("vnosKoce(): entering...");

        boolean isValid = true;

        if (Utils.isSafeEmpty(kocaNaziv)) {
            addFieldError("kocaNaziv", "Koča naziv je obvezen vnos.");
            isValid = false;
        }
        if (Utils.isSafeEmpty(naslov)) {
            addFieldError("naslov", "Naslov je obvezen vnos.");
            isValid = false;
        }
        if (ocena < 1 || ocena > 5) {
            addFieldError("ocena", "Ocena mora biti med 1 do 5.");
            isValid = false;
        }
```

```
        if (!isValid) {
            showKoceForm = true;
            return INPUT;
        }

        KocaDAO kocaDAO = ServiceLocator.getService().getKocaDAO();

        KocaDTO koca = new KocaDTO();
        koca.setKocaId(null);
        koca.setKocaNaziv(kocaNaziv);
        koca.setNaslov(naslov);
        koca.setPostaSt(postaId);
        koca.setTelefon(telefon);
        koca.setOcena(ocena);

        kocaDAO.dodajKoco(koca);

        dobiSeznamKoce();

        LOG.debug("vnosKoce(): leaving...");

        return SUCCESS;
    }

    public Integer getKocaId() {
        return kocaId;
    }
    public void setKocaId(Integer kocaId) {
        this.kocaId = kocaId;
    }
    public String getKocaNaziv() {
        return kocaNaziv;
    }
    public void setKocaNaziv(String kocaNaziv) {
        this.kocaNaziv = kocaNaziv;
    }
    …
    public Collection<PostaTO> getPosta() {
        return posta;
    }
    public void setPosta(Collection<PostaTO> posta) {
        this.posta = posta;
    }
}
```

Lokalne spremenljivke med (*`private String kocaNaziv;`*) se morajo ujemati z imeni polj med (*`<s:textfield label="Koča naziv" name="kocaNaziv"/>`*). Vsaka lokalna spremenljivka mora vsebovati metode getter in setter. S tem dobimo podatke v akcijskem razredu *`Koce`*, ki jih je uporabnik vnesel pri spletnem obrazcu.
Po izvršitvi metode *`execute()`* se vrne rezultat. Če je bil rezultat uspešen (v našem primeru *`return SUCCESS;`*), se pogleda v konfiguracijsko datoteko *`struts.xml`* in poišče, kar se ujema s to vrstico (*`<result name="success">/kocaDodana.jsp</result>`*). To povzroči skok na spletno stran *`kocaDodana.jsp`*.

Če pri vnosu nismo vnesli podatka pri vnosno polje **Koča naziv**, dodamo opis napake (`addFieldError("kocaNaziv", "Koča naziv je obvezen vnos.");`) in preusmerimo na ponoven vnos podatkov iz spletnega obrazca (`return INPUT;`). S tem povzročimo opozorilno napako pri polju **Koča naziv**, kot to prikazuje slika 17.



Slika 17: Opozorilna napaka zaradi manjkajočih podatkov

Če se je zgodila nepričakovana izjema (`throws Exception`), krmilnik ugotovi, da se ujema z vrstico (`<exception-mapping exception="java.lang.Exception" result="error"/>`) in rezultat z vrstico (`<result name="error">/napaka.jsp </result>`), kar povzroči skok na spletno stran `napaka.jsp` z opisom napake.

3.4.3 Nivo uporabniškega vmesnika

Nivo uporabniškega vmesnika je odgovoren za predstavitev in upoštevanje akcij s strani končnega uporabnika. Implementirali smo 20 JSP strani. Pri zasnovi uporabniškega vmesnika je bilo vodilo uporabniška prijaznost. Želeli smo ustvariti enostaven in hiter vmesnik z enostavno obliko, ki bo zasnovan na osnovnih prvinah oblikovanja uporabniških vmesnikov. Ti temelji so organizacija (jasna in dosledna vsebina strukture), varčevanje (narediti čim več s čim manj potezami) in komunikacija (vsebinska predstavitev se ujema s sposobnostmi uporabnika).

3.5 Faza 4 – Namestitev in testiranje

Namestitev in testiranje je zadnja faza pri razvoju po metodologiji RUP. Med in po končanem razvoju spletne aplikacije smo izvajali različne testne scenarije.

Izvajali smo pozitivne in negativne testne scenarije. Pozitivni scenariji so tisti, pri katerih ne simuliramo napake, zato se mora spletna aplikacija izvesti brezhibno. Negativni scenariji pa so tisti, pri katerih ustvarimo oz. simuliramo napako. Negativni scenariji so bili opravljeni

predvsem tam, kjer je uvedena kontrola vnosa, datumov in numeričnih podatkov. S tem skušamo spletno aplikacijo privesti v takšno stanje, da nam javi napako (npr.: vpišemo napačno uporabniško ime ali geslo, manjkajoče podatke pri rezervaciji apartmaja…). Pri testiranju spletne aplikacije smo naleteli na nekaj manjših napak, ki smo jih sproti uspešno odpravili.

V spletno aplikacijo smo vgradili spremljevalec dogodkov obnašanja sistema z uporabo orodja Apache Logging. V vsako metodo smo vstavili preproste izpisne stavke na ključne položaje v naši kodi in v primeru nepričakovane izjeme izpis opisne napake. Pri Apache Logging-u je treba določiti globalno konfiguracijsko datoteko, kjer bo zapisoval dogodke. S tem smo si olajšali iskanje vzrokov povzročitve napak in lažje odkrivanje mesta, kjer je program obtičal.

Namestitev paketa z Java izvršilno kodo in ostalimi datotekami smo izvedli s pomočjo orodja NetBeans IDE. V naši spletni aplikaciji smo napisali dva ločena projekta: projekt za poslovno logiko in projekt za spletno aplikacijo. NetBeans IDE je pri postopku prevajanja samodejno zapakiral izvršilno kodo in ostale datoteke v arhivsko datoteko EAR. Arhivska datoteka EAR je v osnovi ZIP datoteka z metapodatkovno datoteko, v kateri sta JAR in WAR datoteka. Razvojno orodje NetBeans IDE je poskrbel tudi za prenos arhivske datoteke k aplikacijskemu strežniku JBoss. Ročnega posega pri izdelavi arhivske datoteke in namestitve skoraj ni bilo, saj je razvojno orodje NetBeans IDE poskrbelo za vse.

4. Opis uporabe spletne aplikacije za rezervacije smučarskih apartmajev

V tem poglavju bomo opisali uporabo delovanja spletne aplikacije ter povedali nekaj o prednostih in slabostih spletne aplikacije.

4.1 Namen spletne aplikacije

Zaradi sodobnega načina življenja je potencialnim obiskovalcem koč in smučarskih središč spletni način prijave na zasluženi dopust veliko prijaznejši in dostopnejši. Spletni obiskovalci lahko z uporabo naše aplikacije preprosto rezervirajo apartmaje na kateremkoli razpisanem kraju, ki mu ga ta aplikacija ponuja ob pogoju, če je izbran termin prost.

Spletna aplikacija za rezervacijo smučarskega apartmaja je namenjena vsem uporabnikom, ki želijo rezervirati smučarske apartmaje. Predpogoj za rezervacijo apartmaja je registracija in prijava v spletno aplikacijo, s katerim se dostopa do vseh funkcij aplikacije. V nasprotnem primeru si uporabnik lahko samo ogleda opis vseh apartmajev.

4.2 Prednosti spletne aplikacije

Prednosti našega spletne aplikacije so predvsem:

- **Razširjenost spletnih brskalnikov**
  Danes je v večino operacijskih sistemov na voljo vgrajen spletni brskalnik, v nasprotnem primeru pa ga lahko hitro in enostavno namestimo. Sodobni spletni brskalniki so na voljo brezplačno in so zato dostopni vsakomur. Uporabnikom različnih platform je omogočen enakovreden dostop do istih informacij in aplikacij.
- **Enostavna uporaba spletnih brskalnikov**
  Spletni brskalniki so enostavni za uporabo. V osnovi ponujajo funkcije, kot so: navigacija med stranmi, shranjevanje spletnih naslovov, tiskanje vsebine in pregled zgodovine uporabe.
- **Dostopnost spletnih aplikacij**
  Prednost spletnih aplikacij je, da se aplikacija nahaja na enem samem mestu in je dostopna poljubnemu številu uporabnikov. Uporabnik lahko uporablja spletno aplikacijo na katerikoli lokaciji, ki ima dostop do interneta.
- **Namestitev spletne aplikacije na eno lokacijo**
  Ker se spletna aplikacija nahaja na eni sami lokaciji, opravimo vse nadgradnje, vzdrževanje, arhiviranje podatkov in odpravo napak na enem mestu. S tem zmanjšamo velike stroške. S tem dosežemo, da je tako uporabniku vedno na voljo zadnja verzija programa.

- **Globalizacija podatkov**
  Veliko podjetij uporablja namizne aplikacije, ki so nameščene na oddaljenih lokacijah, in lokalne podatkovne baze. Te baze se ob določenem časovnem intervalu sinhronizirajo. Danes je ključnega pomena ažurnost in dostopnost podatkov, ki jih takšne aplikacije ne morejo zagotoviti. Ta problem v celoti odpravijo spletne aplikacije, ki za svoje delovanje uporabljajo eno centralno podatkovno bazo.

4.3 Slabosti spletne aplikacije

Na žalost imajo spletne aplikacije, tudi nekaj pomanjkljivosti. Te so za nekatere uporabnike in razvijalce bolj, za druge manj pomembne.

Slabosti naše aplikacije so:

- **Varnost**
  Čeprav se varnost vsak dan izboljšuje, so spletne aplikacije še vedno bolj izpostavljene neavtoriziranemu dostopu kot namizne aplikacije. Ko ponudimo aplikacijo uporabnikom preko interneta, jo nehote ponudimo tudi osebam, ki bi utegnile zaradi zabave ali drugih zlih namenov onesposobiti strežnik ali zlorabiti občutljive podatke. S tem povzročajo nedostopnost aplikacije za vse njene uporabnike.
- **Zahtevnejši razvoj**
  Kljub prednostim vzdrževanja le ene namestitve spletne aplikacije pa zahteva razvoj in vzdrževanje spletnih aplikacij veliko več znanja in truda. Logiko aplikacije in uporabniške vmesnike, ki jih je pri razvoju namiznih aplikacij razmeroma enostavno implementirati, ni mogoče vedno implementirati v takšni meri, da bi bile spletne aplikacije sposobne nadomestiti namizne aplikacije. Danes so na voljo tehnologije, ki močno olajšajo razvoj spletnih aplikacij, a enostavnost razvoja namiznih aplikacij daleč presega razvoj spletnih aplikacij.
- **Obremenjenost strežnika**
  Strežnik, na katerem teče spletna aplikacija, prevzame vso obdelavo podatkov in procesiranje logike. Pri večjem številu uporabnikov ta pogosto ni več sposoben izvajati vseh zahtev uporabnikov. Rešitev je nakup zelo zmogljivih in dragih sistemov kot tudi zakup širokopasovnih povezav, ki omogočajo dovolj veliko prepustnost podatkov.

4.4 Uporaba spletne aplikacije za rezervacijo smučarskega apartmaja

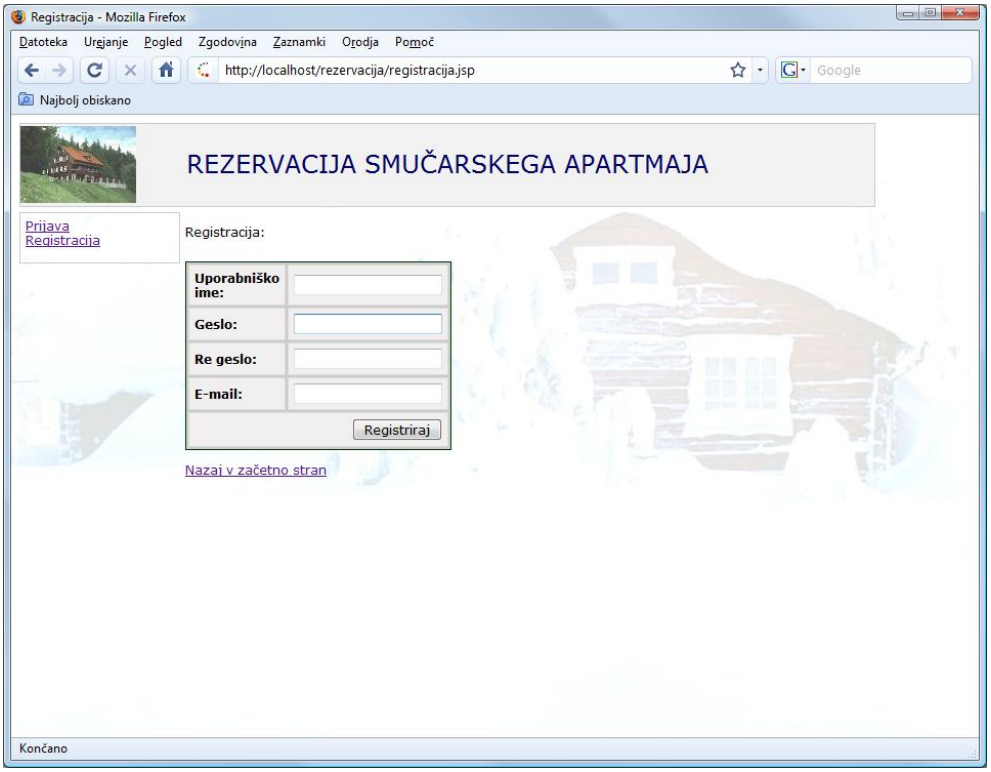
Spletne aplikacije lahko uporabljajo vsi potencialni gostje koč, ki jih zanima ponudba. Če se še niso registrirali v spletni aplikaciji, imajo na voljo samo vpogled ponudbe. Registrirani uporabniki, ki so že nekoč obiskali ponujene apartmaje ali jih imajo namen še kdaj obiskati, imajo večjo funkcionalnost. Tu je še skrbnik spletne aplikacije, ki skrbi za ažurnost podatkov in ima pristojnost potrditve/zavrnitve rezervacij.

4.4.1 Vloga spletne aplikacije za gosta in registriranega uporabnika

4.4.1.1 Registracija uporabnika

S postopkom registracije uporabnik pridobi polni dostop do vseh funkcij spletne aplikacije, ki jih omogoča. Predvsem sta to funkciji:

- rezervacija apartmaja in
- pregled svojih rezervacij.

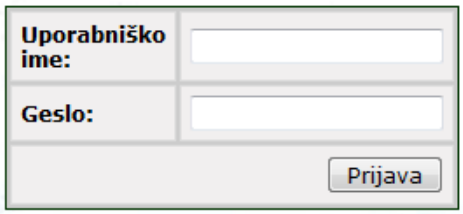


Slika 18: Registracija uporabnikov

Registracija se izvede tako, da se klikne povezavo **registracija**, s čimer se odpre obrazec za izpolnjevanje podatkov. V vnosno polje se vpiše uporabniško ime, geslo in veljaven elektronski naslov. Registracijo potrdimo na gumb **registriraj**. Na vpisani elektronski naslov uporabnik prejme elektronsko sporočilo z vsebovano aktivacijsko povezavo za aktiviranje računa za prijavo v spletno aplikacijo. Po potrditvi aktivacijske povezave se lahko uporabnik prijavi v spletno aplikacijo (slika 18).

4.4.1.2 Prijava

Pri kliku na povezavo **prijava** se uporabniku prikaže vnosna maska za prijavo v spletno aplikacijo. V vnosno polje **uporabniško ime** in **geslo** vpiše podatke, s katerimi se je uporabnik registriral v spletni aplikaciji. Prijava se potrdi s klikom na gumb **prijava**. V primeru, da so vpisani podatki pri prijavi napačni, sistem opozori uporabnika z obvestilom o napaki. V nasprotnem primeru je uporabnik prijavljen (slika 19).



Slika 19: Obrazec za prijavo v sistem

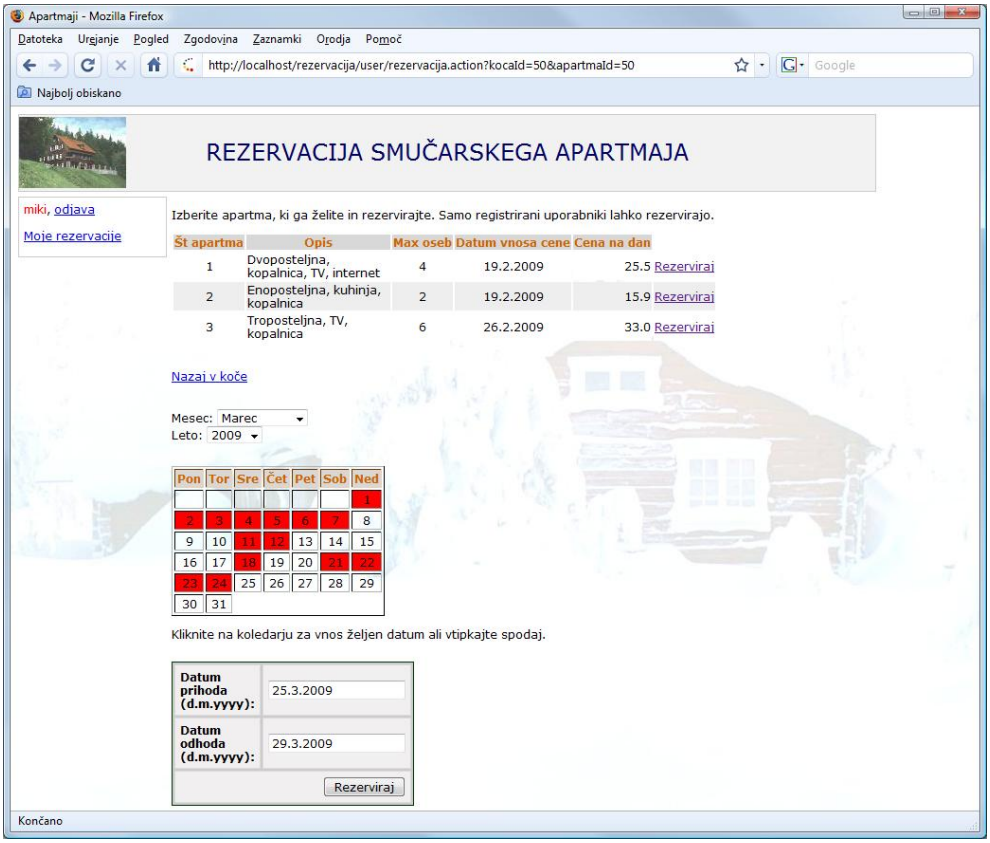
4.4.1.3 Rezervacija apartmaja

Naziv	Naslov	Posta	Kraj	Ocena	Telefon	
Koca kanin	Pesje 15	1000	Ljubljana	★★★★☆	01 345 5554	Apartmaji
Koca TikTak	Hudobra 55	2000	Maribor	★★★☆☆	02 4336 120	Apartmaji
Koca Dremlje	Trebnje 56	1000	Ljubljana	★★★★★	01 345 5554	Apartmaji
Koca vicec	Brzina 22	8253	Artice	★☆☆☆☆	08 3214 5567	Apartmaji
Koca Zame	Hrusja 33	2000	Maribor	★★☆☆☆	02 4456 333	Apartmaji

Slika 20: Seznam smučarskih koč

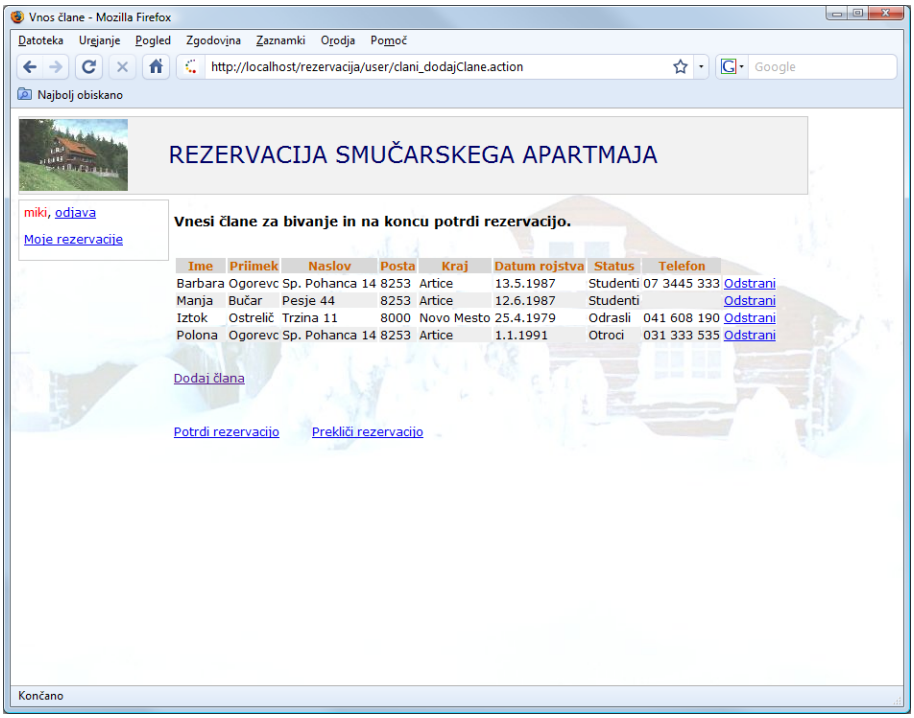
Če želi uporabnik rezervirati apartma za določen datum, si najprej izbere kočo na poljubnem kraju (slika 20). Nato si glede na kapaciteto izbere apartma, ki jo ta ponuja, torej glede na število članov, ki bodo bivali v apartmaju. Po kliku na povezavo **rezerviraj** se odpre mesečni koledar in spletni obrazec za vnos datuma prihoda in odhoda. Mesečni koledar je lahko obarvan z rdečim poljem za določen termin, kar pomeni, da je nekdo že prej rezerviral in zasedel zaželeni termin. Uporabnik lahko vnaša datum ročno pri vnosnem obrazcu ali pa s

klikom na dan v mesečnem koledarju. Pri kliku na dan v mesečnem koledarju sistem samodejno vnese datum v vnosno polje **datum prihoda**, ob ponovnem kliku na koledarju se vnese datum v vnosno polje **datum odhoda**. Uporabnik nato potrdi gumb **rezerviraj**. Če sistem ugotovi, da je na vneseni datum prost termin, sistem rezervira apartma za ta termin, v nasprotnem primeru pa sistem opozori uporabnika z obvestilom o napaki (slika 21).



Slika 21: Rezervacija smučarskega apartmaja

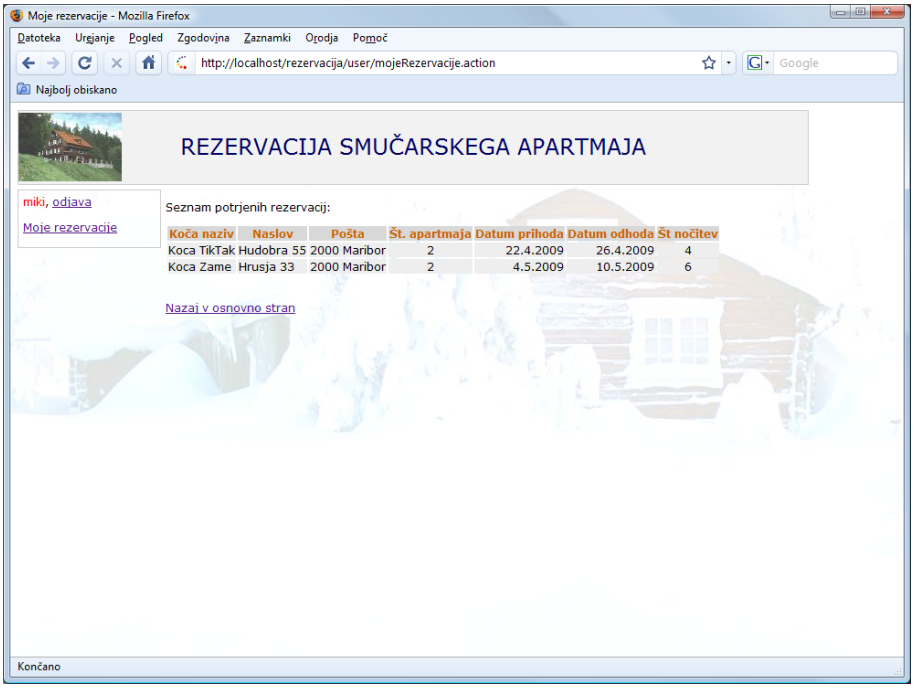
Po kliku gumba **rezerviraj** se odpre nova stran, kjer uporabnik vnaša osebne podatke vseh članov, ki bodo bivali v določenem rezerviranem apartmaju. Ko uporabnik vnese vse člane, klikne na povezavo **potrdi rezervacijo**. Sistem preveri, če je število vnesenih članov enako kapaciteti bivanja v apartmaju in zaključi rezervacijo, v nasprotnem primeru pa sistem posreduje obvestilo z napako. Če želi uporabnik iz kakršnegakoli razloga preklicati ali prekiniti rezervacijo, ima možnost klika na povezavo **prekliči rezervacijo**. S tem se rezervacija apartmaja razveljavi (slika 22).



Slika 22: Vnos članov za bivanje v apartmaju

4.4.1.4 Moje rezervacije

Po prijavi ima uporabnik možnost izbire **moje rezervacije**, v katerem ima pregled seznama vseh potrjenih rezervacij (slika 23).



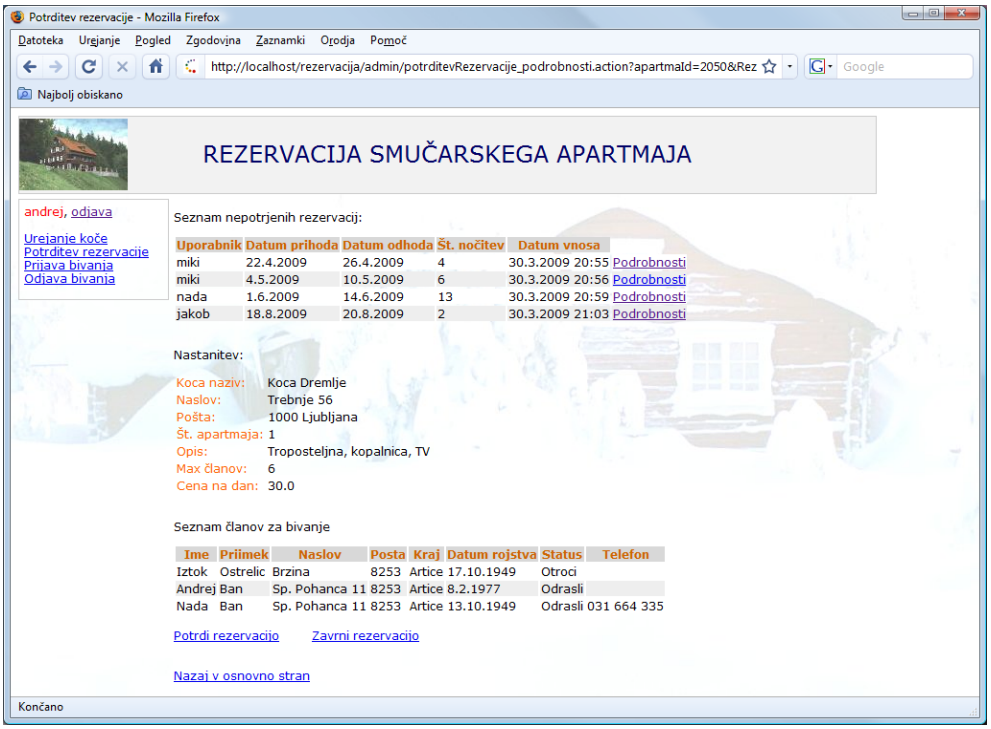
Slika 23: Seznam mojih rezervacij

4.4.2 Vloga spletne aplikacije za uporabnika skrbnika (lastnika)

Skrbnik ima poleg pregleda seznama koč in apartmajev še potrditev rezervacij, prijavo bivanja v apartmaju, odjavo bivanja iz apartmaja ter urejanje podatkov koče in apartmajev.

4.4.2.1 Potrditev rezervacije

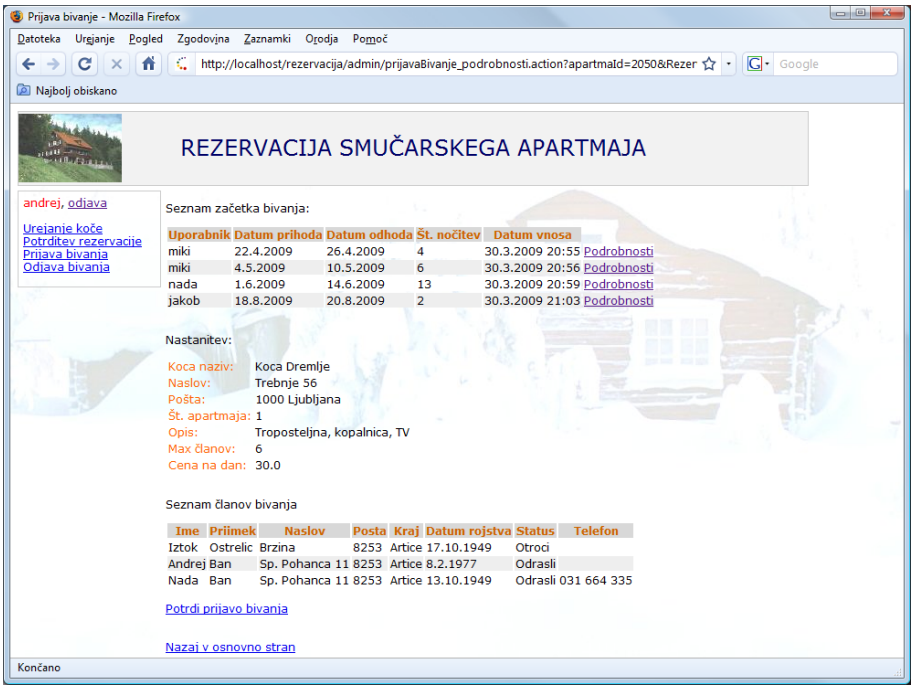
Na sliki 24 ima skrbnik spletne aplikacije seznam nepotrjenih rezervacij, ki so jih pred kratkim rezervirali uporabniki. Skrbnik spletne aplikacije ima možnost, da za vsakega uporabnika pregleda podrobnosti o podatkih rezervacij in članov, nato pa te rezervacije s klikom na povezavo **potrdi rezervacijo**. Potrdi ali s klikom na povezavo **zavrni rezervacijo** zavrne. To je recimo v primeru, ko je bila rezervacija pomanjkljivo izpolnjena.



Slika 24: Potrditev rezervacije

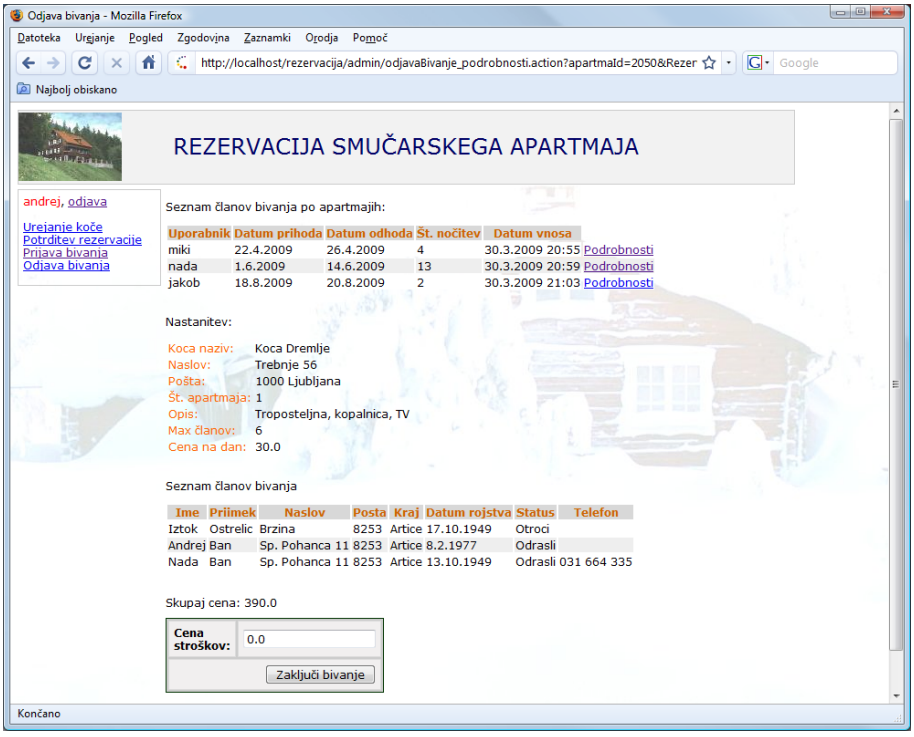
4.4.2.2 Prijava bivanja

Ko nastopi datum začetka bivanja v apartmaju, odpre skrbnik povezavo **prijava bivanja** in najde seznam tistih uporabnikov, ki čakajo vselitev v apartma. Skrbnik si izbere določenega uporabnika iz seznama, pogleda podrobnosti o rezervaciji in seznam članov. Če se podatki ujemajo, se lahko bivanje v apartmaju začne brez težav, zato skrbnik preda ključe uporabniku in klikne na povezavo **potrdi prijavo bivanja**. S tem potrdi, da člani od tega trenutka dalje bivajo v apartmaju (slika 25).



Slika 25: Prijava bivanja

4.4.2.3 Odjava bivanja

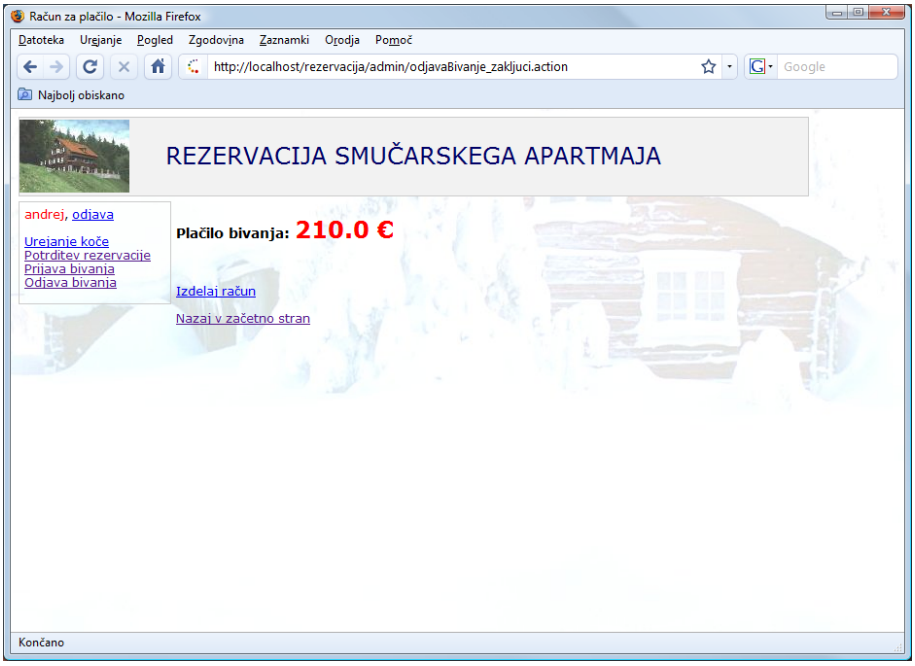


Slika 26: Odjava bivanja

Ko nastopi datum izselitve iz apartmaja, skrbnik klikne na povezavo **odjava bivanja**. Odpre se stran, kjer ima seznam vseh uporabnikov, ki trenutno bivajo v apartmajih. Skrbnik si izbere

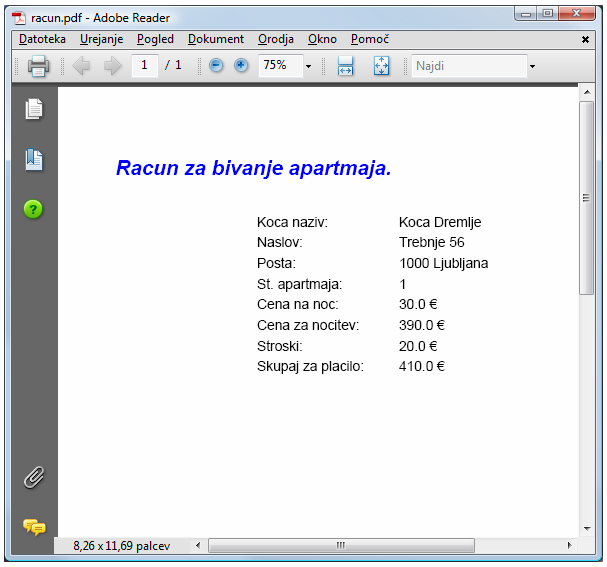
tistega uporabnika, ki se mora izseliti iz apartmaja in poravnati račun za bivanje v apartmaju. Sistem mu izračuna skupno ceno na podlagi števila nočitev in ceno posamezne nočitve. Poleg tega mu ponudi obrazec za vnos cene stroškov, ki so nastali med časom bivanja. Skrbnik vnese ceno stroškov bivanja (ali pa ne) in zaključi z gumbom **zaključi bivanje** (slika 26).

Po kliku na gumb **zaključi bivanje** se odpre nova stran, kjer se izpiše skupna cena za plačilo bivanja apartmaja (slika 27).



Slika 27: Stran za prikaz skupne cene za plačilo

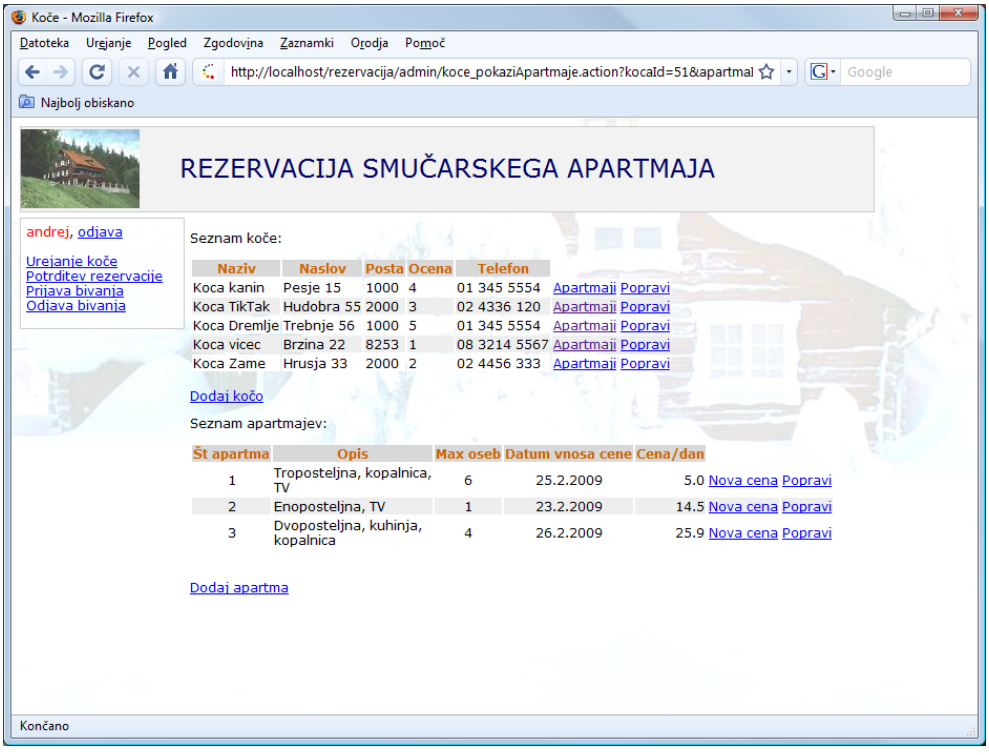
Poleg tega ima skrbnik možnost shraniti račun v PDF datoteki ali pa še natisniti na papir. To lahko naredi s klikom na povezavo **izdelaj račun** (slika 28).



Slika 28: Izdelava računa v datoteki PDF

4.4.2.4 Urejanje koče

Pri urejanju koče ima skrbnik možnost dodajati nove koče in apartmaje ali popraviti podatke o že obstoječih kočah in apartmajih. Če želi skrbnik povečati ceno prenočišča za bivanje v apartmaju, lahko to stori tako, da ob izbranem apartmaju klikne na povezavo **nova cena**. Po kliku se dinamično odpre obrazec za vnos nove cene (slika 29).



Slika 29: Urejanje koč in apartmajev

5. Sklepne ugotovitve

Cilj diplomskega dela je bil opisati korake pri izdelavi spletne aplikacije v izbrani tehnologiji z uporabo metodologije RUP ter razviti konkretno aplikacijo. Kot je razvidno, smo spletno aplikacijo uspešno razvili.

Pri delu so nam bila v pomoč razna orodja. Izbor trinivojske arhitekture je bila pravilna odločitev, saj omogoča spremembe v vseh delih programske kode. Njena prednost je tudi hiter preklop aplikacije na drugo podatkovno bazo ali podatkovni vir. V našem primeru je spletna aplikacija pisana za podatkovno bazo Oracle. Za preklop na drugo bazo bi bilo potrebno spremeniti samo sloj dostopa do podatkov, kar pomeni, da poslovna logika ostane ista.

Jedro naše rešitve so strežniška javanska zrna, v katerih je izvedena poslovna logika v skladu s pravili, ki jih postavlja specifikacija. Razvijalec se ne ukvarja z nizkonivojskim sistemskim programiranjem, mrežnimi protokoli ali transakcijami, saj vse te storitve zagotavlja sama arhitektura strežniških zrn. Tehnologija Struts 2 temelji na modelu MVC, kar nam je olajšalo izdelavo spletne strani, avtorizacijo uporabnika in komunikacijo s strežniškim javanskim zrnom. Če ne bi uporabili te tehnologije, bi morali vložiti bistveno več dela v pisanje programske kode, varnost dostopa do spletne strani in prikaz seznama podatkov.

Pri razvoju spletne aplikacije smo utrdili znanje programiranja v programskem jeziku Java, ki smo ga pridobili na fakulteti za računalništvo in informatiko ter na delovnem mestu. V času razvoja spletne aplikacije smo se postopoma naučili uporabljati ogrodje Struts 2, ki ga prej nismo poznali. Ogrodje Struts 2 je zelo obsežno, zato zaradi omejenega časa nismo mogli uporabiti vseh funkcionalnosti. To ponuja dodatne možnosti za izboljšavo spletne aplikacije z uporabo trikov in pasti.

Na temo diplomske naloge se da še veliko narediti. Spletna aplikacija se recimo lahko izpopolni na naslednji način:

- izdelava letnega poročila o številu rezervacij apartmajev po kočah,
- iskanje podatkov o rezervacijah v preteklosti,
- spletno aplikacijo bi lahko uporabljalo več različnih administratorjev, ki bi imeli na skrbi upravljanje rezervacij apartmajev na določenih lokacijah.

6. Viri

[1] (1999) Aleš Živkovič: Arhitektura Java EE. Dostopno na: http://cot.uni-mb.si/cotl/99jesenzima/jee.html

[2] Donald Brown, Chad Michael David: *Struts2 in Action*. Manning Publications, 2007.

[3] (2007) Luka Pavlič: Poenostavitve v Java EE. Dostopno na: http://164.8.251.136:8080/lp/pages/sl/publics/ots07/ots07.pdf

[4] Michael Sikora: *EJB 3 Developer Guide*. Packt Publishing, 2008.

[5] (2006) Rok Rupnik: Prosojnice predavanj 2005/06 predmeta *Osnove informacijskih sistemov*

[6] (2009) Aplikacijski strežnik JBoss. Dostopno na: http://en.wikipedia.org/wiki/Jboss

[7] (2009) IBM Rational Unified Process. Dostopno na: http://en.wikipedia.org/wiki/Rup

[8] (2009) NetBeans IDE. Dostopno na: http://en.wikipedia.org/wiki/NetBeans

[9] (2009) NetBeans IDE features. Dostopno na: http://www.netbeans.com/features

[10] (2009) Oracle Database. Dostopno na: http://en.wikipedia.org/wiki/Oracle_database

[11] (2009) Oracle JDeveloper. Dostopno na: http://en.wikipedia.org/wiki/Jdeveloper

[12] (2009) RMI-IIOP. Dostopno na: http://en.wikipedia.org/wiki/RMI-IIOP

[13] (2009) Visual Paradigm for UML. Dostopno na: http://en.wikipedia.org/wiki/Visual_Paradigm_for_UML