Univerzitet u Sarajevu
Elektrotehnički fakultet u Sarajevu
Odsjek za računarstvo i informatiku

Studentski dom

Objektno orjentisana analiza i dizajn

Dženana Huseinspahić, Esma Karahodža
i Dženeta Kudumović

Sadržaj

1 Opis projekta

Na svim većim univerzitetima širom svijeta veliki broj studenata živi u studentskim domovima. Namjena ovog softvera je da studentima olakša izvršavanje svakodnevnih aktivnosti u domu i da uposlenicima pomogne da što efikasnije izvršavaju neke od resursno zahtjevnih poslova i studentskih servisa. Na taj način se automatiziraju odreeni procesi koji oduzimaju dosta vremena studentima i osoblju koje ih izvršava.

Funkcionalnosti

- Prijem studenata u studentski dom – popunjavanje aplikacije, kontrola dokumentacije, odobravanje zahtjeva
- Održavanje informacija o studenatima - lični podaci i osnovne informacije vezane za servise u studentskom domu
- Dnevna rezervacija obroka u kuhinji od strane studenata
- Prijava zahtjeva za tekućim održavanjem studentskih apartmana i zajedničkih prostorija
- Upravljanje zahtjevima za održavanje – dodjela zadataka tehničkom osoblju, evidentiranje završetka poslova i izvještavanje

Akteri

- Student
- Uposlenik uprave
- Šef kuhinje
- Šef odjela tehničkog održavanja

2 Klase

Student

- atributi
  - imeIPrezime: String
  - JMBG: String
  - adresaStanovanja: String
  - fakultet: String
  - godinaStudiranja: int
  - brojTelefona: int
  - brojSobe: int
  - email: String
  - brojIndeksa: int
  - brojBonova: int
- metode
  - <<create>>Student(in s:String, in jmbg:String, in a:String, in f:String, in g:int, in br:int, in brS:int, in e:String, in i:int, in b:int)

StudentskiDom

- atributi
  - studenti: List¡Student¿
- metode
  - <<create>>studentskiDom()
  - prijava(in student:Student): void
  - odjava(in student:Student): void

PrijavaObroka

- atributi
  - student: Student
  - rucak: Boolean
  - vecera: Boolean
  - zaPonijet: Boolean
- metode
  - <<create>>prijavaObroka(in s:Student)
  - smanjiBonove(): void
  - dodajPrijavu(in s:Student): PrijavaObroka

PrijavaUDom

- atributi
  - student: Student
  - vrijemePrijave: Date
- metode
  - <<create>>prijavaUDom(in s:Student, in d:Date)

PrijavaKvara

- atributi
  - student: Student
  - tipKvara: enum
  - opisKvara: String
  - vrijemeKvara: Date
- metode
  - <<create>>prijavaKvara(in s:Student, in t:enum, in o:String, in v:Date)
  - dodajPrijavu(in s:Student): PrijavaKvara

UposlenikDoma

- interface
- atributi
  - imeIPrezime: String

SefKuhinje

- izvedena klasa iz klase UposlenikDoma
- atributi
  - obroci: List<PrijavaObroka>
- metode
  - <<create>>sefKuhinje()
  - odobriPrijave(): void

UposlenikUprave

- izvedena klasa iz klase UposlenikDoma
- atributi
  - prijave: List<PrijavaUDom>
- metode
  - <<create>>uposlenikUprave(s: String)
  - odobriPrijavuUDom(prijava: PrijavaUDom): Boolean
  - promijeniPodatkeOStudentu(student: Student): void
  - unesiStudenta(student: Student): void

SefTehnickogOdrzavanja

- izvedena klasa iz klase UposlenikDoma
- atributi
  - radnici: List<Radnik>
  - kvarovi: List<PrijavaKvara>
- metode
  - <<create>>sefTehnickogOdrzavanja()
  - pronadjiRadnika(prijava: PrijavaKvara): PrijavaKvara

Radnik

- izvedena klasa iz klase SefTehnickogOdrzavanja
- atributi
  - Usluga: enum
  - brojTelefona: int
- metode
  - <<create>>radnik(usluga: enum, brojTelefona: int)

3 Solid principi

3.1 Single Responsibility Principe

Princip S zahtijeva da svaka klasa ima samo jednu odgovornost, odnosno da klasa vrši samo jedan tip akcija kako ne bi ovisila o prevelikom broju konkretnih implementacija.

U ovom projektu princip S je ispoštovan jer svaka klasa vrši samo jednu vrstu akcija, npr. postoje tri različite klase za različite prijave umjesto jedne koja bi slala podatke svim klasama.

3.2 Open/Closed Principle

Princip O zahtijeva da klasa koja koristi neku drugu klasu ne treba biti modificirana pri uvoenju novih funkcionalnosti, ili pri potrebi za mijenjanjem druge klase.

Vidimo da za sve klase na klasnom dijagramu vrijedi da možemo mijenjati okruženje oko klase bez promjene same klase. Npr. na našem klasnom dijagramu nijedna klasa me zavisi od neke druge klase na način da bi promjenom te druge klase morali modificirati i prvu.

3.3 Liskov Substitution Principle

Princip L zahtijeva da nasljeivanje bude ispravno implementirano, odnosno da je na svim mjestima na kojima se koristi osnovni objekat moguće iskoristiti i izvedeni objekat a da takvo nešto ima smisla.

Liskov princip je ovdje zadovoljen jer se sve naslijeene klase mogu zamijeniti svojim osnovnim tipom, npr. klasu šefKuhinje možemo zamijeniti klasom Uposlenik Doma bez da stvorimo konflikt.

3.4 Interface Segregation Principle

Princip I zahtijeva da i svi interfejsi zadovoljavaju princip S, odnosno da svaki interfejs obavlja samo jednu vrstu akcija.

Klasni dijagram sadrži samo jedan interfejs, UposlenikDoma. On obavlja samo jednu akciju a to je da povezuje tri vrste uposlenika doma. Zbog toga je i ovaj princip zadovoljen.

3.5 Dependency Inversion Principle

Princip D zahtijeva da pri nasljeivanju od strane više klasa bazna klasa uvijek bude apstraktna. Razlog za ovo je što je teško koordinisati veliki broj naslijeenih klasa i konkretnu baznu klasu ukoliko ista nije apstraktna, a da pritom kod bude čitak i jednostavan za razumijevanje.

Ovaj projekat ne sadrži nijednu baznu klasu koja nije apstraktna i samim time je ovaj princim automatski ispunjen.

4 MVC