UNIVERZITET
METROPOLITAN
BEOGRAD

Jesenji semestar 2020/2021

Game Engine

SE201
Uvod u softversko inženjerstvo

Projektna dokumentacija

Student:
Aleksa Cekić 4173

Asistent:
Jovana Jović

Sadržaj

```java
package implementation;

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.*;

class SistemTest {

    @Test
    void prikazGui() {
        System.out.println("Provera prikazivanja GUI!");
        String unos = "Gui se sada prikazuje";
        Sistem sis = Sistem.getsistem();
        String result = sis.prikazGui();
        assertEquals(unos, result);
    }
}
```


Uvod

Cilj ovog dokumenta je da opiše funkcionalnosti procesa sistema za izradu video igara, tj. **Game Engina.** Ovaj projektni zadatak baviće se definisanjem osnovnih funkcionalnih i nefunkcionalnih zahteva koje softver mora da zadovolji kako bi se smatrao uspešnim. U dokumentu će biti detaljno opisan rad sistema o kome je reč. Za bolje razumevanje problema sa kojim se susrećemo kako bi projekat bio realizovan biće objašnjeni putem korišćenja dijagrama (Klasni, Slučaj Korišćenja, Sekvencijalni, itd...) koji bi trebalo detaljnije da objasne način i zamisao funkcionisanja sistema. Za potrebe izrade dijagrama koristiće se alat PowerDesigner. Prikaz interfejsa za korišćenje biće realna aplikacija koja već radi u produkciji i pokazuje rezultate. Korisnici ovog sistema su programeri koji rade na kreiranju neke video igre kao i inženjeri koji rade na tom sistemu.

Svrha dokumenta

Svrha dokumenta je da jasno opiše moguće funkcionalnosti i mogućnosti softvera. Dokument se sastoji od samog uvoda gde se daje kratak opis cilja za kreiranje ovog sistema. Nakon toga, dokument će se sastojati od opisa sistema njegovim mogućnostima, detalje zahteva u kojima su opisani funkcionalni i

nefunkcionalni zahtevi. Dokument će se sastojati i od specifikacije dizajna u okviru kog se nalaze dijagrami slučajeva korišćenja, klasni dijagram i svih mogućih sekvencijalnih dijagrama koji su vezani za rad sistema za dizajniranje video igre. Nakon toga, dokument će dati opis funkcionisanja sistema u vidu slika.

Opis

Ovaj game engine je dizajniran da omogući korisniku da kreira razne video igre u prostom, ali dosta opširnom okruženju. Ovaj sistem omogućava korisnicima da se loguju na sistem što im omogućava da koriste razne alate namenjene za razvoj igara. Inženjer ima mogućnost da održava aplikaciju, i da je redovno ažurira. Ažuriranje aplikacije ne sme da ometa u radu sistema.

Funkcije sistema

- **Autentikacija korisnika sistema**
- **Biranje programskog jezika**
- **Pisanje funkcionalnosti**
- **Pravljenje nivoa**
- **Grafičko modeliranje**
- **Animacije modela**
- **Dizajniranje modela i likova**
- **Održavanje aplikacije**

Korisnici sistema

Postoji dva tipa korisnika a to su korisnici koji koriste ovaj sistem za izradu video igara kao i inženjeri koji održavaju aplikaciju.

Opis sistema za korisnika

Korisnik sistema treba da se uloguje na sam sistem. Nakon toga može da pristupi ostalim funkcijama. Odobreno mu je da bira jezik u kome će da radi, da piše funkcionalnosti, da animira, da kreira modele i nivoe. Ukoliko nije odabrao jezik a želi da piše neku funkcionalnost, sistem će ga upozoriti na to da nije izabran ni jedan jezik i da bi korisnik dobio pristup pisanju funkcionalnosti on mora da izabere neki od ponuđenih jezika.

Opis sistema za inženjera

Inženjer ima mogućnost da održava sistem i da samim tim vrši redovna ažuriranja sistema i aplikacije.

Metodologija izrade

Metodologija koju ćemo koristiti za izradu ovog sistema je **inkrementalni metod razvoja.** Za izbor ovog metoda je zaslužna činjenica da će aplikacija uvek biti nadograđivana najnovijim tehnologijama, updejtovima, i slično. Samim tim što je aplikacija napravljena za kreiranje drugih aplikacija ona će morati da se usavršava uvek, model zasnovan na upotrebi komponenata zbog toga što će se pri kreiranju neke video igre generalno koristiti većina unapred definisanih komponenti.

Specifikacija zahteva

Funkcionalni i nefunkcionalni zahtevi

Zahtev	Opis
1.	**Korisnik**
1.1	**Programski jezik** Korisnik ima mogućnost da bira programski jezik koji će da koristi pri izradi video igre.
1.2.	**Pisanje funkcionalnosti** Korisnik u svom izabranom jeziku piše funkcionalnost igre.
1.3	**Grafičko 3D modelovanje** Korisnik će modelovati karaktere, modele, bilo šta što bi uticalo na grafički dizajn igre.
1.3.1	**Animacije** Korisnik će praviti animacije za određene rekvizite.
1.3.2	**Dizajn** Korisnik će korišćenjem grafičkog dizajna moći dizajnirati razne rekvizite.
1.4	**Pravljenje nivoa** Korisnik ima mogućnost da pravi razne nivoe za svoju igru.
1.4.1	**Dizajn nivoa** Korisnik će korišćenjem grafičkog dizajna dizajnirati nivo koji je napravio.

2.	**Inženjer**
2.1	**Održavanje aplikacije** Inženjer ima mogućnost da održava aplikaciju I da je ažurira.

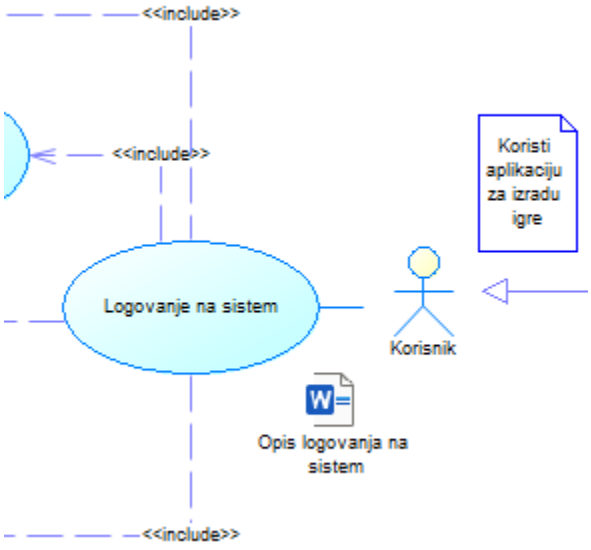
Tabela 1 Funkcionalni zahtevi korisnika

Zahtev	Opis
1.	**Raspoloživost/dostupnost** Aplikacija mora biti dostupna u bilo kom trenutku izazov redovnog održavanja sistema.
2.	**Bezbednost** Aplikacija mora da stiti licne podatke korisnika. Ako korisnik promasi lozinku vise od 3 puta zahteva se verfikacioni kod koji je poslat na e-mail adresu. Ako je korisnik neaktivan nakon nekog vremena aplikacija ce ga izlogovati. Aplikacija cuva sve izmenjene podatke nakon nekog kraceg vremena korisnikove neaktivnost.
3.	**Lakoća koriććenja** Sistem mora biti jednostavan za sve korisnike, za korisnike razlicitog znanja. Za pocetnike da bude dobar radi ucenja i upoznavanja, ali i da moze biti dosta funkcionalan za naprednije korisnike.

Tabela 2 Nefunkcionalni zahtevi korisnika

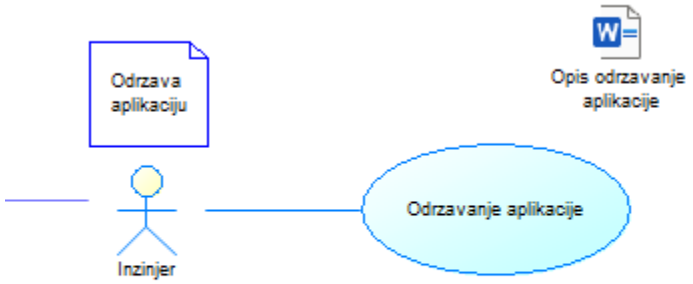
Slučajevi upotrebe

Slučaj upotrebe za logovanje na sistem



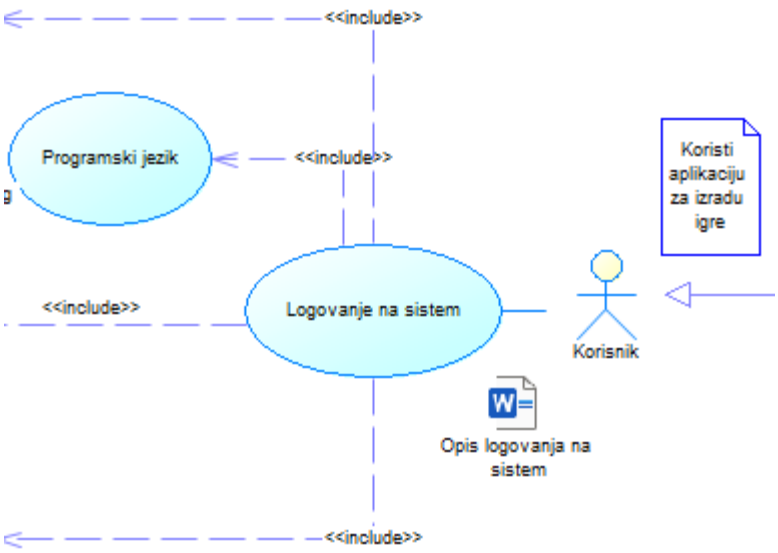
USE CASE	Sistem logovanja
Aktori	Korisnik
Preduslov	Korisnik se registrovao, I postoji u bazi podataka.
Cilj	Pristup sistemu izrade
Okidac	Klik na dugme
Normalni tok	1. Korisnik unosi ime 2. Unosi prezime 3. Loguje se
Alternativni tok	[ALT – 1] 1. Korisnik mora napraviti nalog 2. Registruje se na sistemu 3. Potvrdjuje podatke 4. Verifikuje e-mail adresu 5. Loguje se

Slučaj upotrebe održavanje sistema



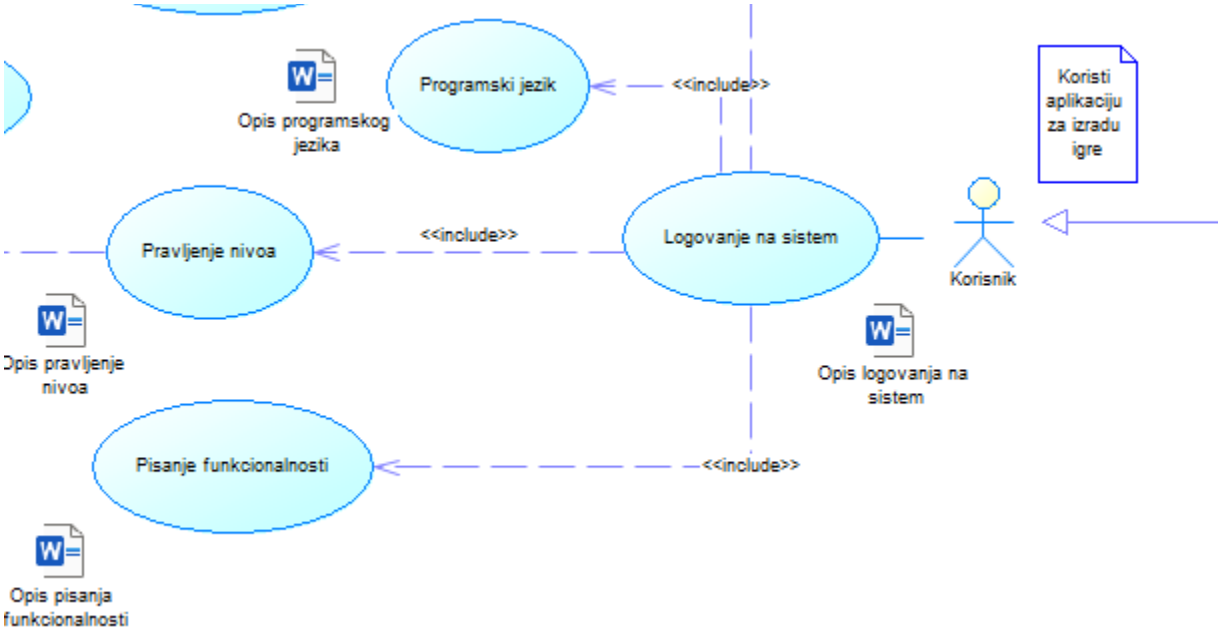
USE CASE	Odrzavanje aplikacije
Aktori	Inzenjer
Preduslov	Inzenjer je ulogovan na sistemu.
Cilj	Drzati aplikaciju “UP-TO-DATE”
Okidac	Uploadovanje updejtovanog koda
Normalni tok	1. pisanje novog koda 2. izmena starog koda 3. dodaje nove izmene 4. pronalazi I uklanja greske 5. uploaduje kod
Alternativni tok	[ALT – 1]buggovi 1. pronaci buggove u programu 2. debuggovati 3. proveriti nove ismene 4. azurira kod

Slučaj upotrebe programskog jezika



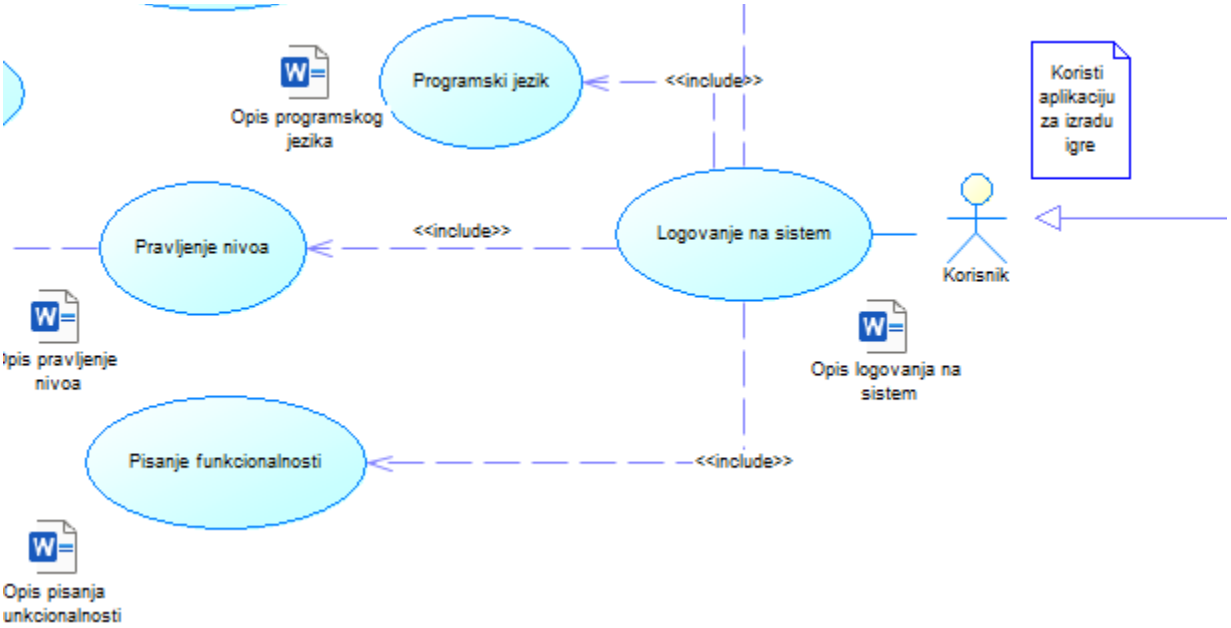
USE CASE	Programski jezik
Aktori	Korisnik
Preduslov	Korisnik je odlucio koji jezik zeli da koristi pri izradi aplikacije, napravio je nalog, ulogovao se.
Cilj	Korisnik mora da izabere jezik u kome pise funkcionalan kod.
Normalni tok	1. Korisnik izabere jezik 2. Pise funkcionalnost na tom jeziku

Slučaj upotrebe pisanja funkcionalnosti



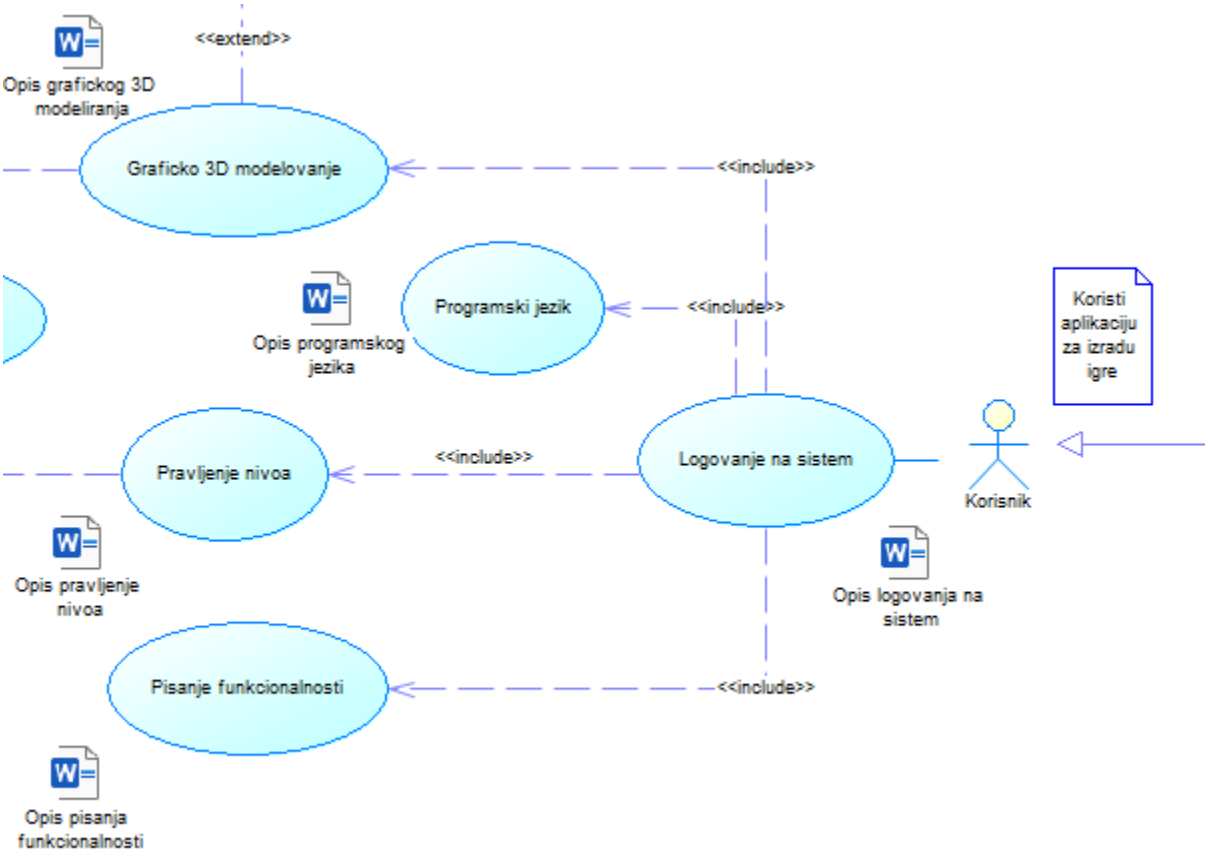
USE CASE	Pisanje funkcionalnosti
Aktori	Korisnik
Preduslov	Korisnik se registrovao, I postoji u bazi podataka.
Cilj	Cela igrica radi onako kako je korisnik napisao njenu funkcionalnost u programskom jeziku koji je on odabrao.
Okidac	Za svaku novu komponentu pravi se dodatni fajl koji je skripta na kome on pise svoj kod.
Normalni tok	1. Korisnik kreira neki entitet 2. Fajl se napravi uz taj entitet/Korisnik sam pravi fajl za funkcionalnost nekog dela igrice 3. Korisnik pise kod u jeziku koji je on odabrao. 4. Cuva sve podatke koje je uneo/izmenio. 5. Proverava da li igra radi kako treba.

Slučaj upotrebe pravljenje nivoa



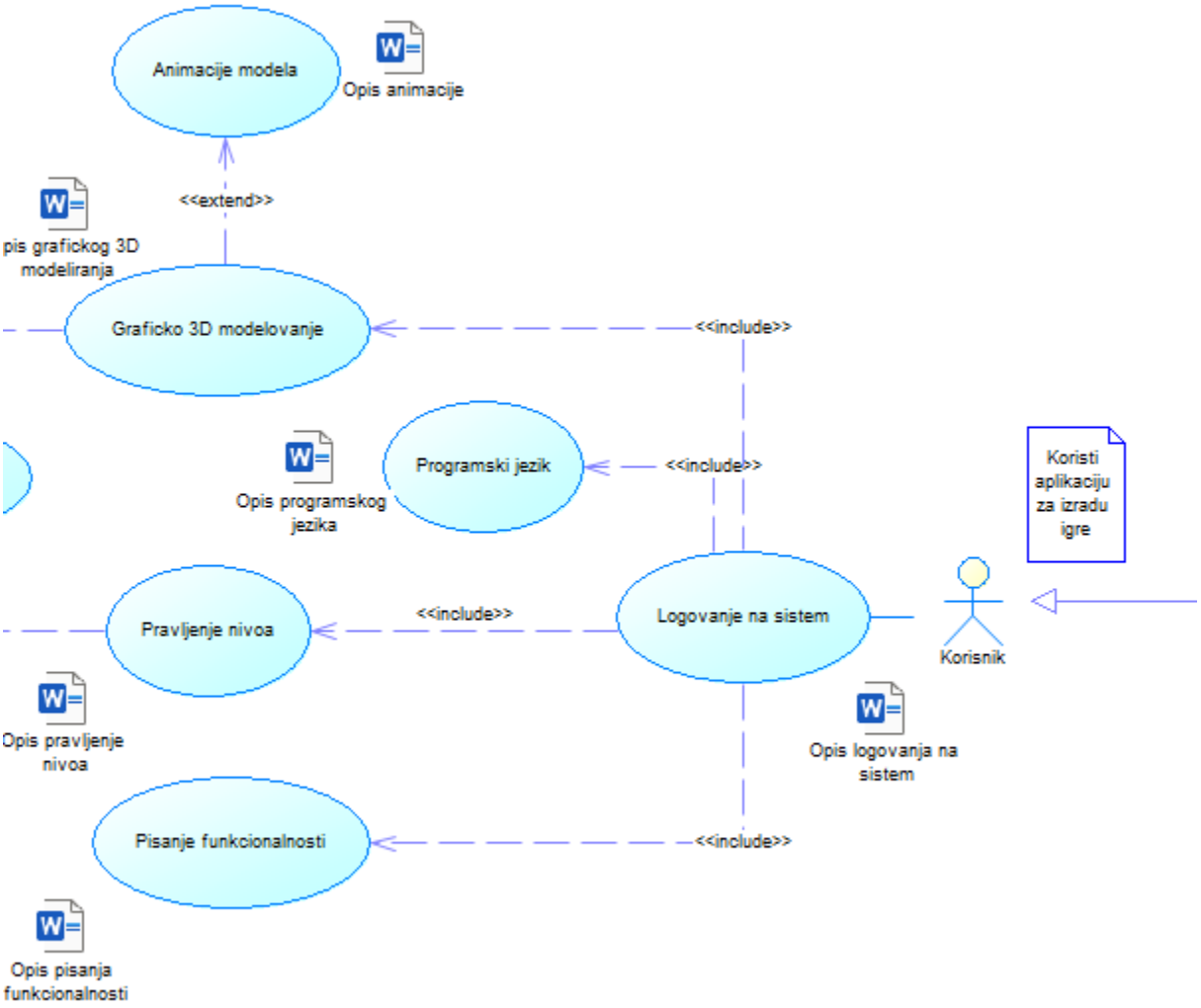
USE CASE	Pravljenje nivoa
Aktori	Korisnik
Preduslov	Korisnik se registrovao, I postoji u bazi podataka.
Cilj	Pravljenje nivoa za video igru.
Okidac	Klik na sekciju za pravljenje nivoa.
Normalni tok	1. Korisnik pravi nivo na okruzenju u sistemu 2. Koristi odredjene rekvizite da bi pravio nivo 3. Cuva podatke na cloudu ili na hard disku.

Slučaj upotrebe grafičkog 3D modelovanja



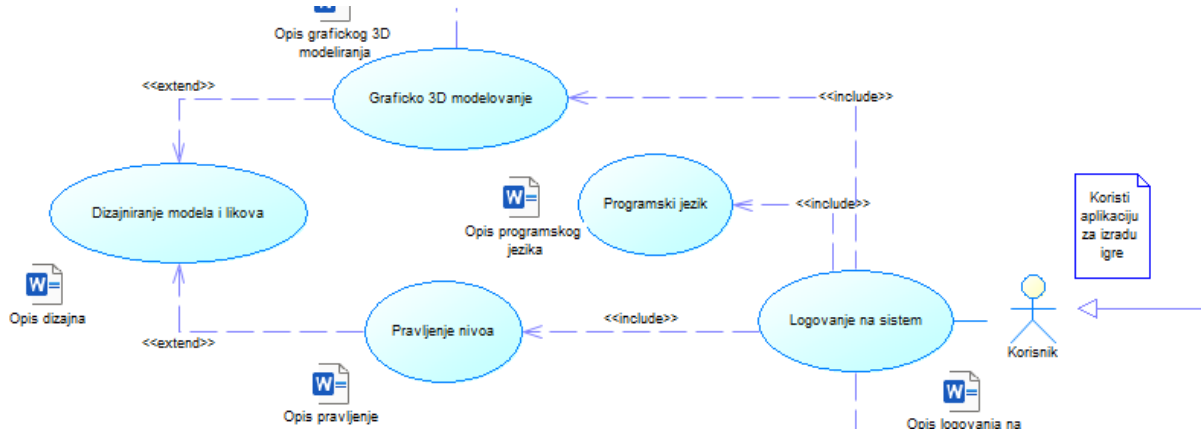
USE CASE	Graficko 3D modeliranje
Aktori	Korisnik
Preduslov	Korisnik je ulogovan na system, I postoji u bazi podataka
Cilj	Praviti rekvizite koji ce se koristiti u video igri koju korisnik zeli da napravi.
Okidac	Klik na dugme za 3D modeliranje
Normalni tok	1. Korisnik bira opciju da kreira 3D objekte 2. Kreira modele na nekom okruzenju 3. Cuva svoje podatke na cloudu ili na hard disku.

Slučaj upotrebe animacije modela



USE CASE	Animacija
Aktori	Korisnik
Preduslov	Korisnik se registovao na I postoji u bazi podataka
Cilj	Korisnik animira entitete sa kojima ce raditi
Okidac	Klik na dugme za animaciju
Normalni tok	1. Korisnik se uloguje na aplikaciju 2. Bira da animira entitete 3. Animira ih preko okruzenja za animaciju

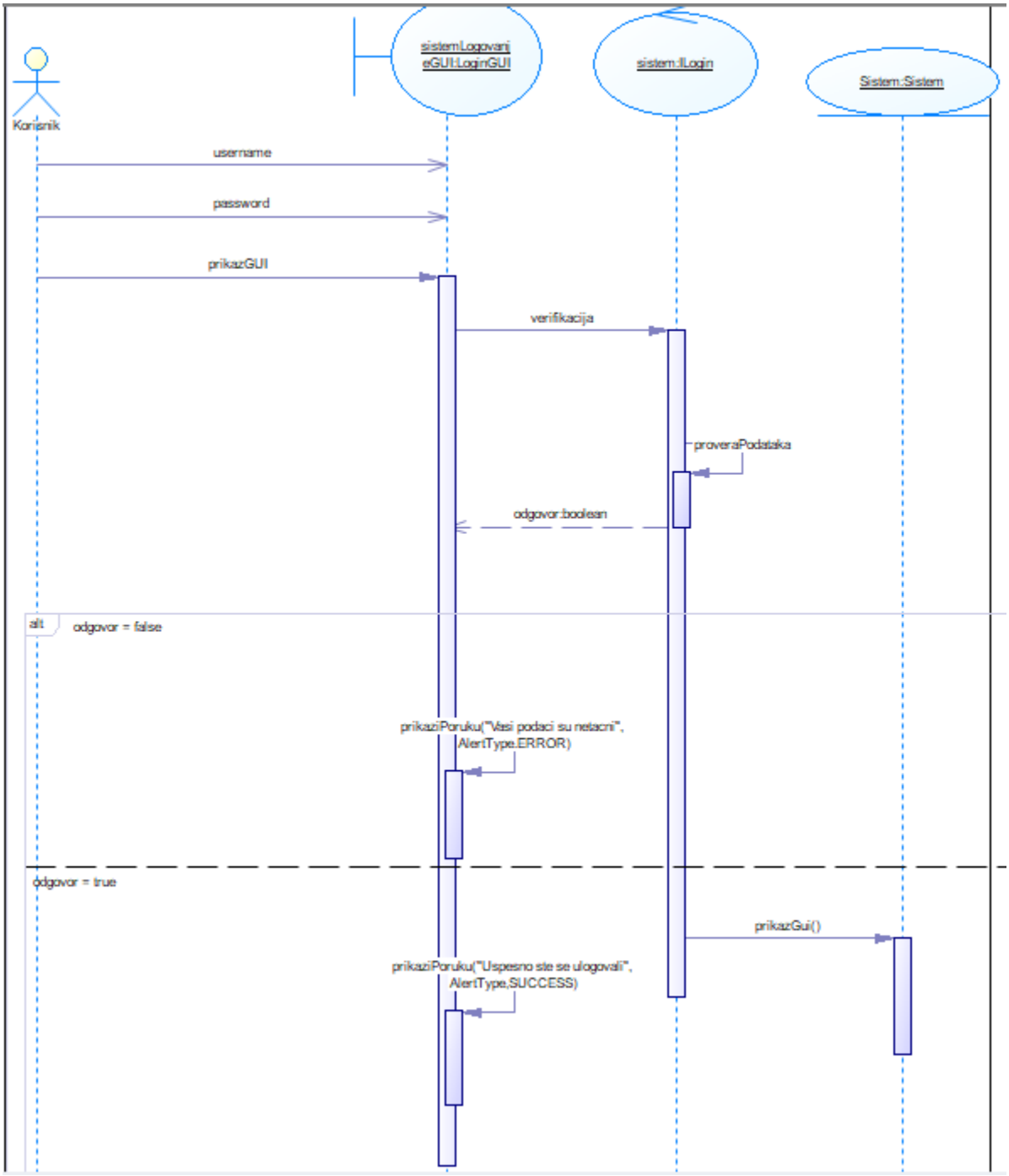
Slučaj upotrebe dizajniranje modela i likova



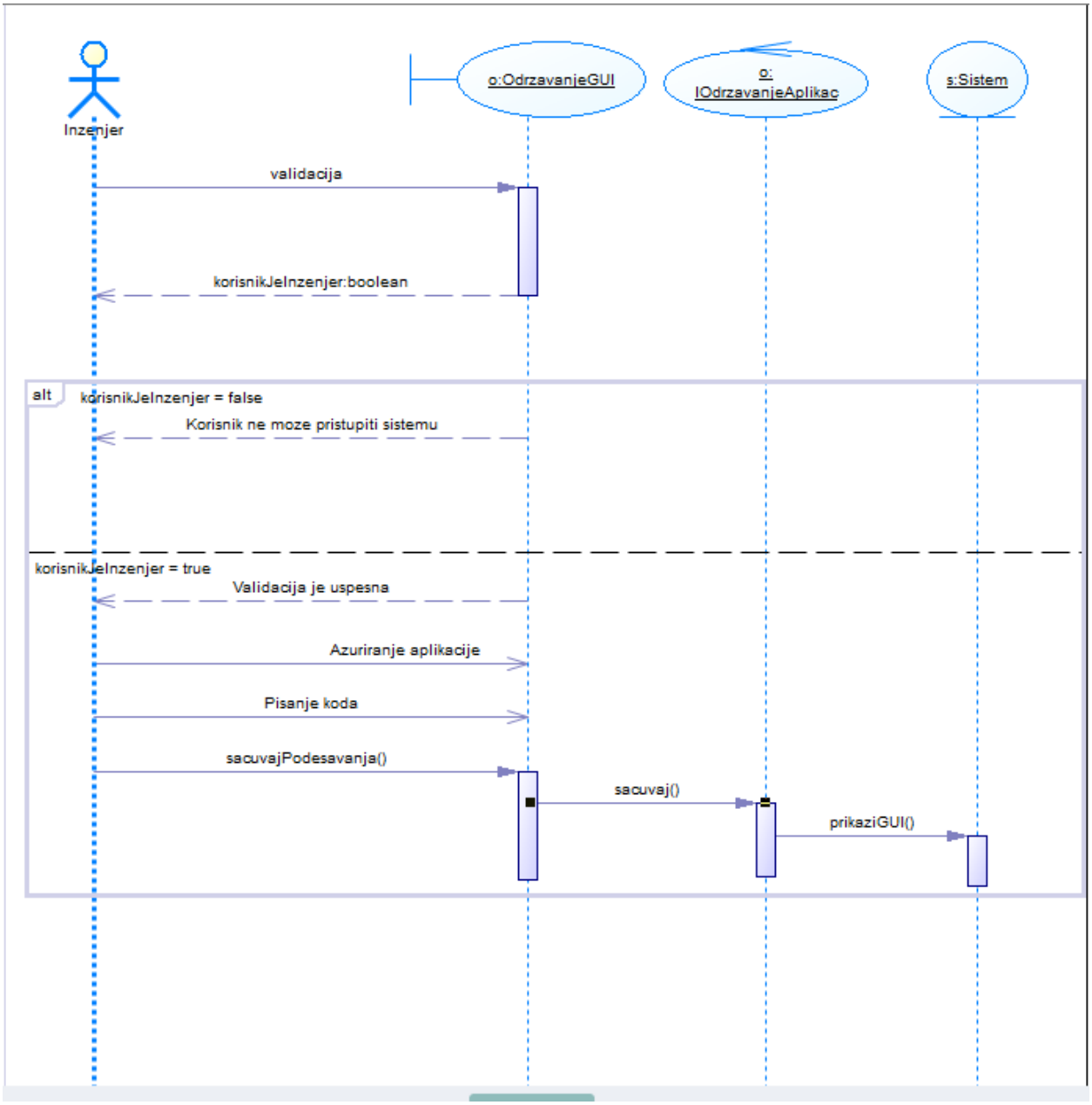
USE CASE	Dizajn
Aktori	Korisnik
Preduslov	Korisnik je ulogovan na system, I postoji u bazi podataka
Cilj	Dizajniranje nivoa, I rekvizita koje je korisnik napravio ili skinuo.
Okidac	Klik na dugme za dizajn.
Normalni tok	1. Korisnik klikom otvara opciju za animaciju 2. Animira odredjene nivoe ili rekvizite preko odredjenog okruzenja.

Dijagrami sekvenci

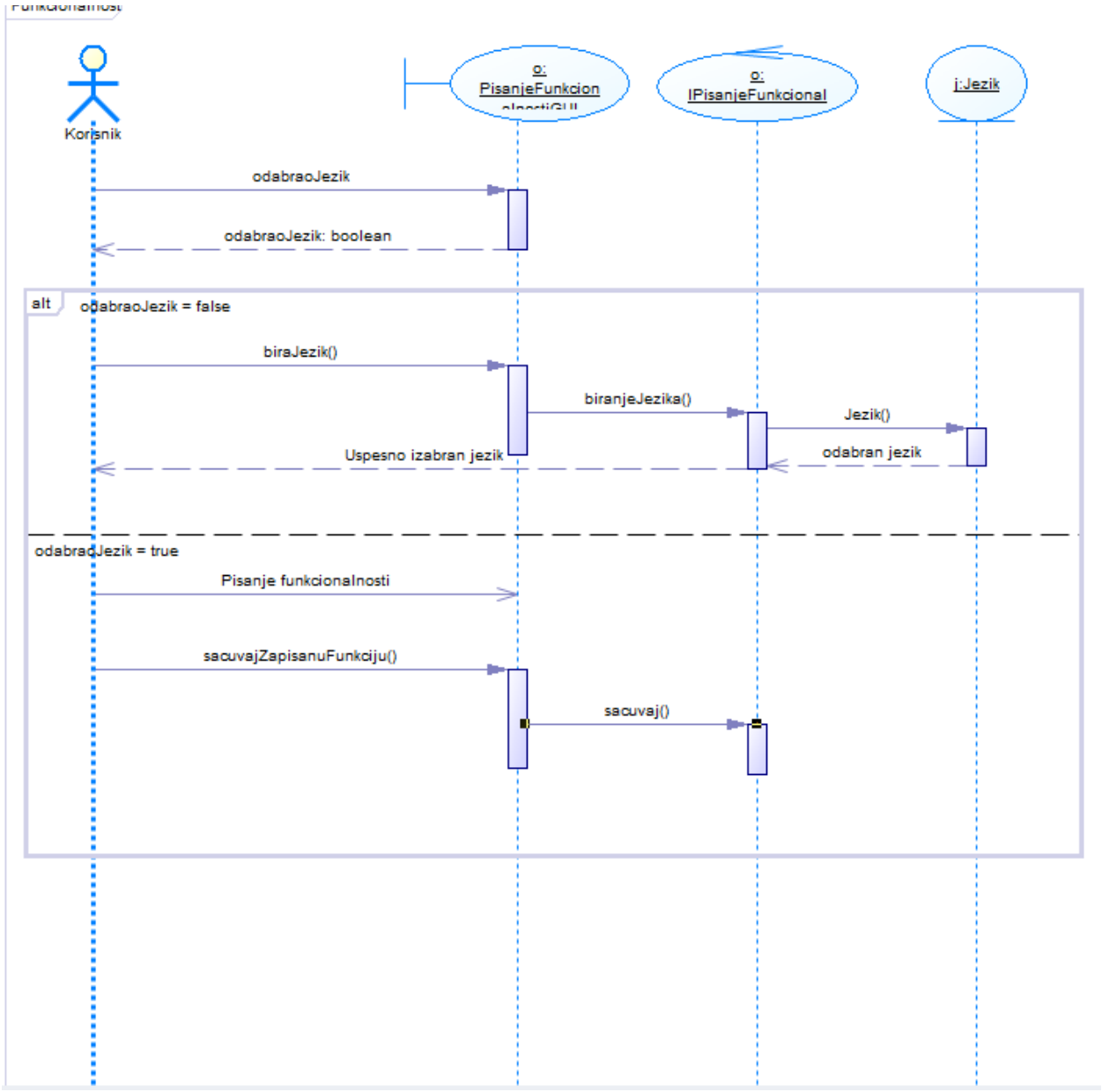
Dijagram logovanja



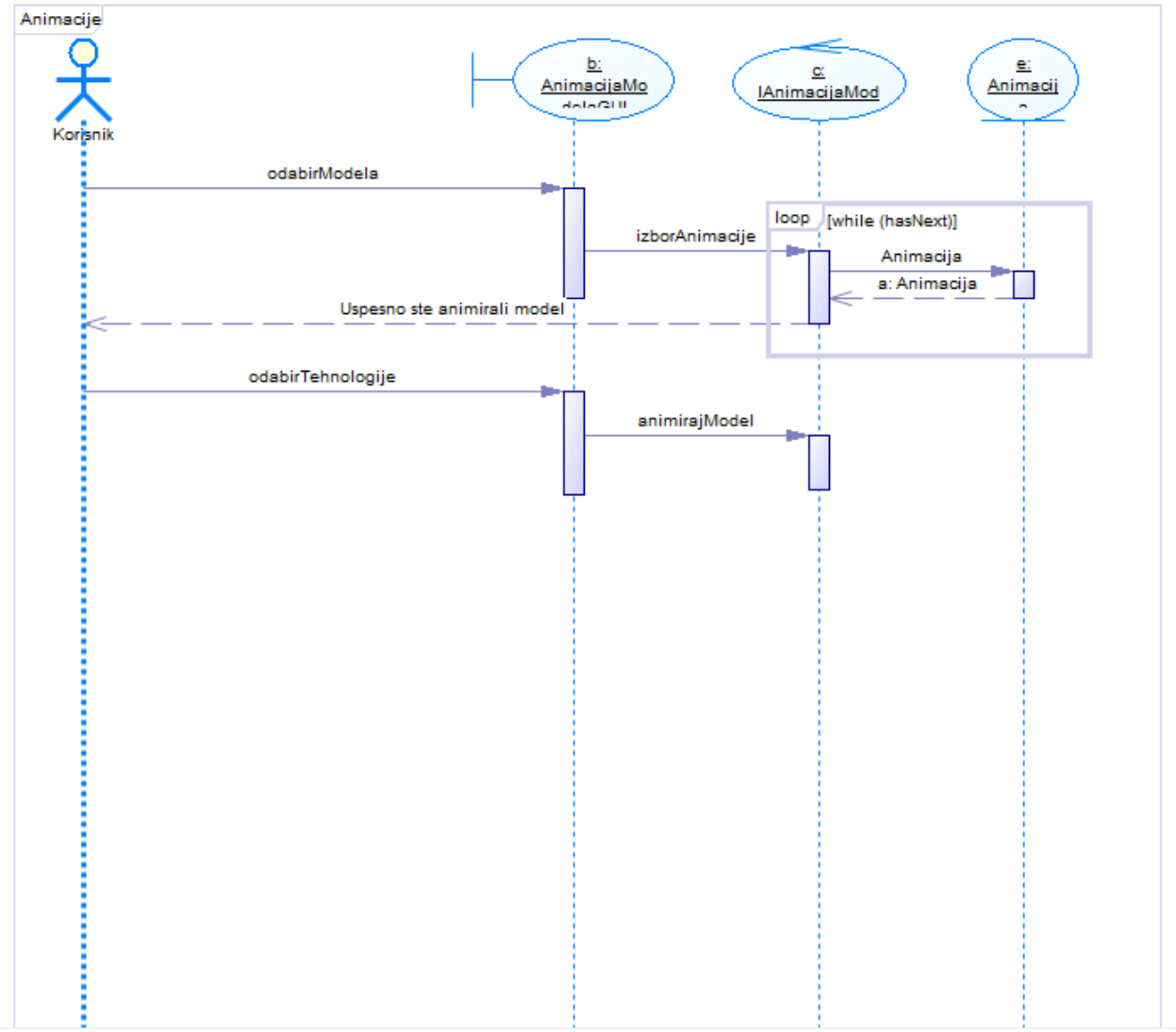
Dijagram održavanje sistema



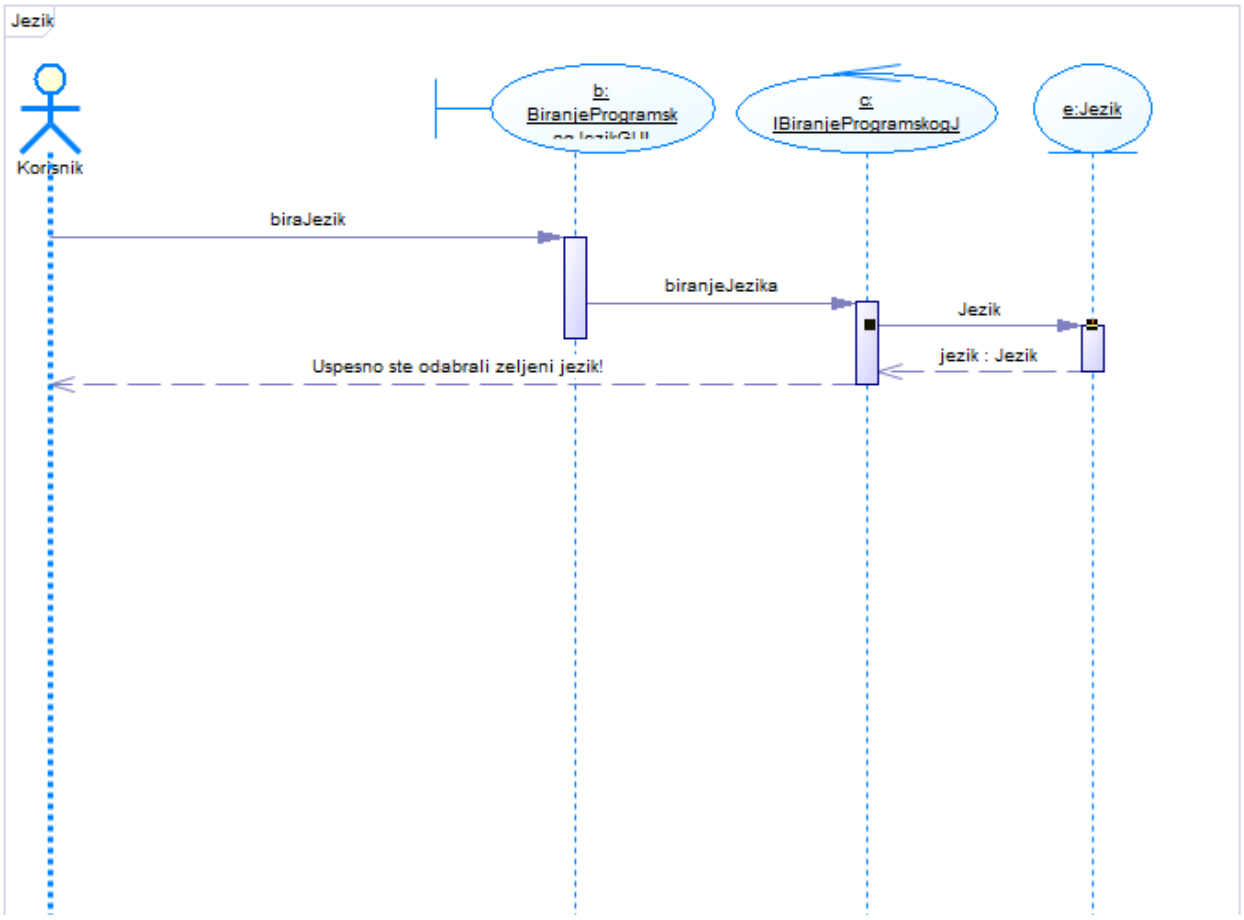
Dijagram pisanja funkcionalnosti



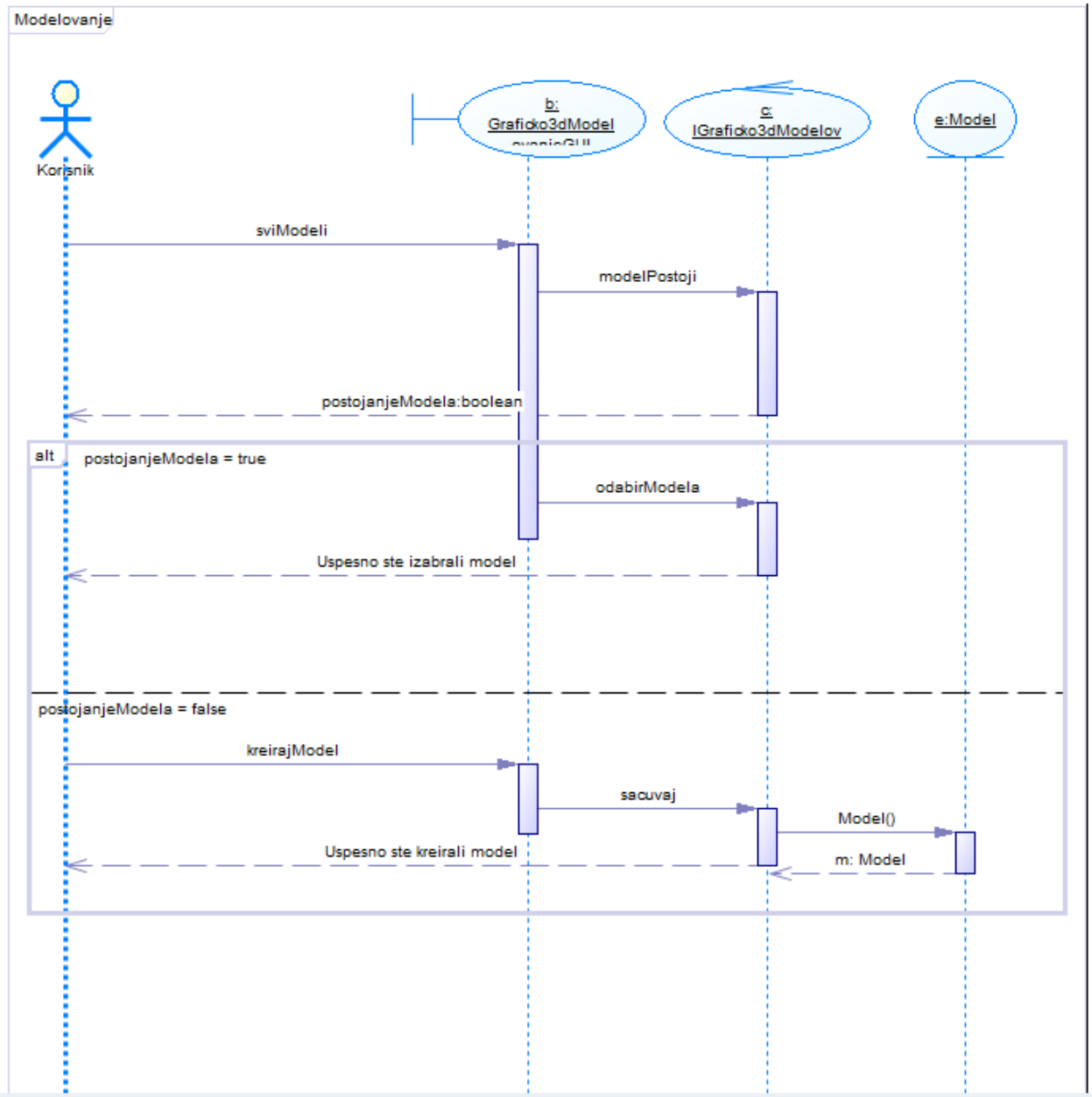
Dijagram animacije modela



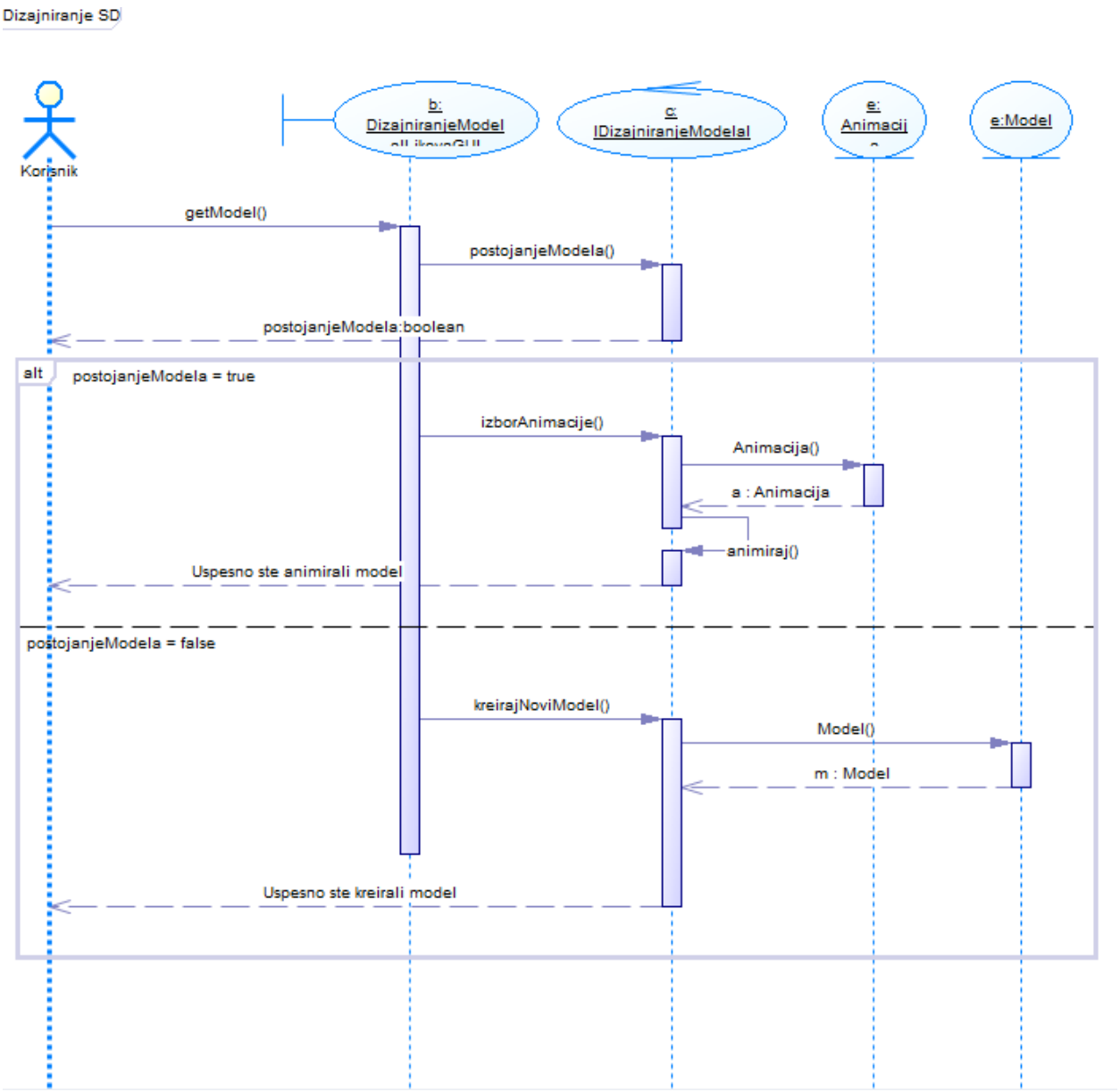
Dijagram odabir jezika



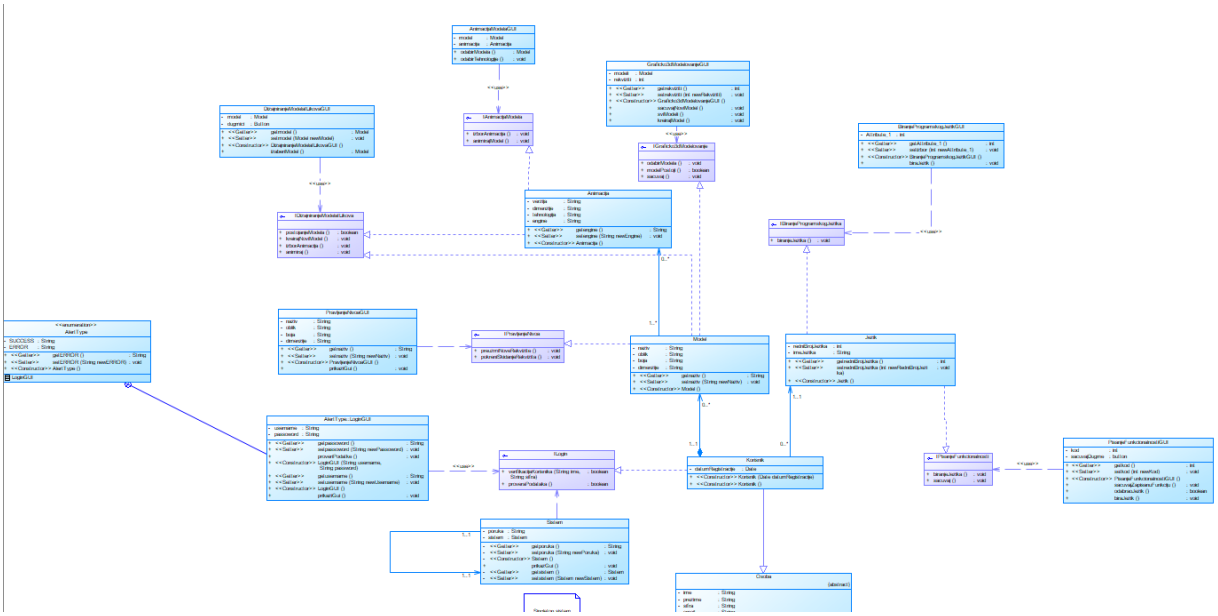
Dijagram grafičkog modelovanja



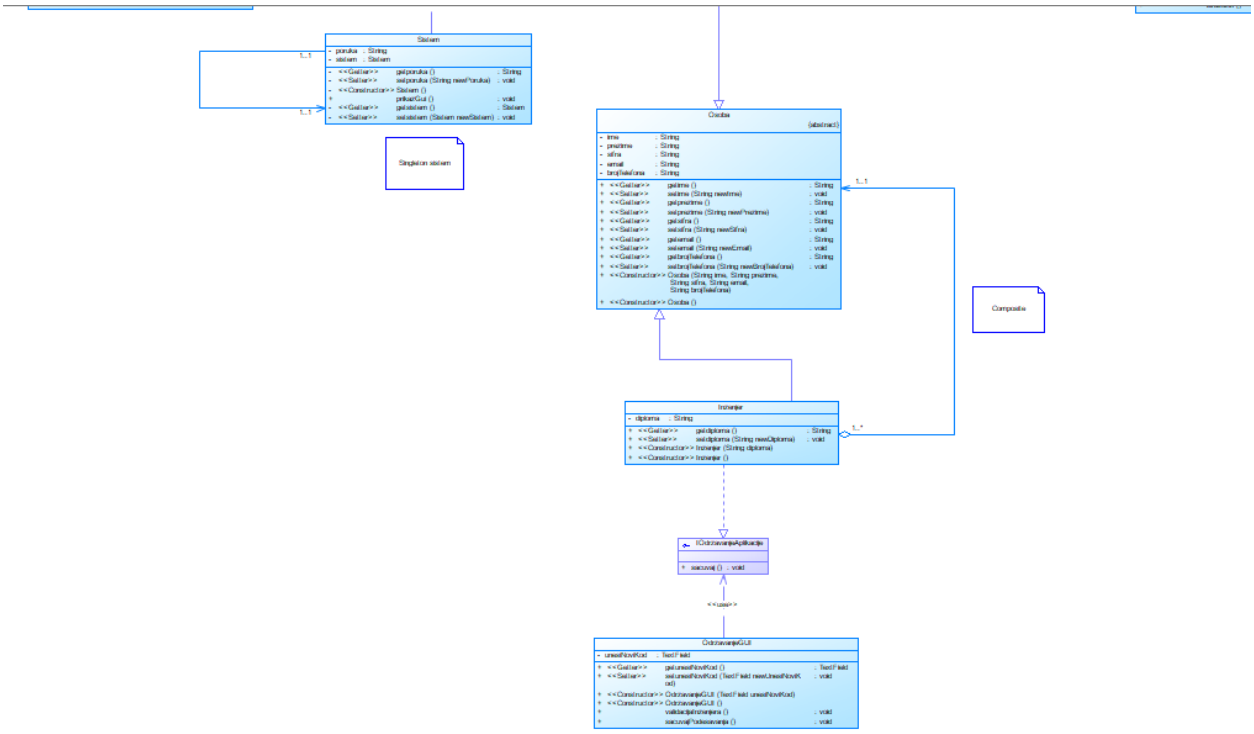
Deteljan dijgram dizajniranje modela



Klasni dijagram

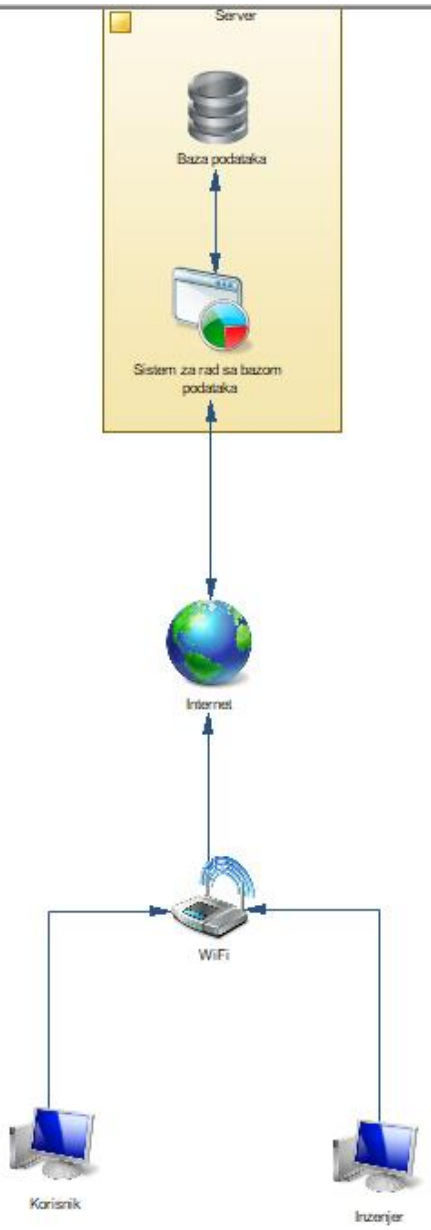


Slika 1 Prvi deo klasnog dijagrama



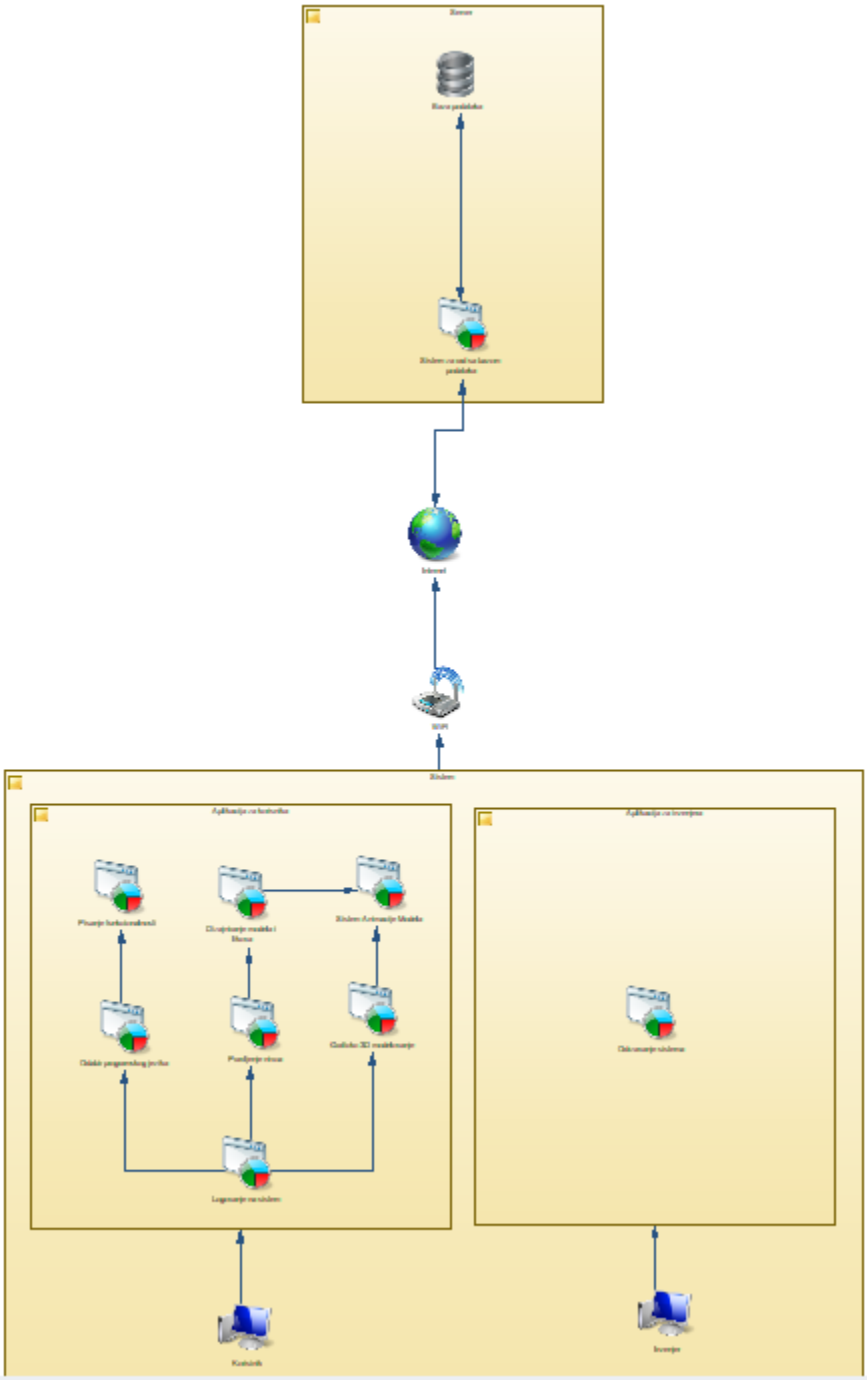
Slika 2 Drugi deo klasnog dijagrama

Arhitektura sistema



Slika 3 Arhitektura sistema

Infrastruktura sistema



Slika 4 Infrastruktura sistema

Testiranje

Testiranje modula proverava da li pojedinačne funkcije i komponente ispravno funkcionišu sa svim očekivanim tipovima ulaza, u skladu sa dizajnom komponente. Jedinično testiranje treba vršiti u kontrolisanom okruženju gde rezultati metoda mogu biti precizno proućen i biti im određena validnost.

Java programski jezik u sebi ima ugrađeno testiranje pod nazivom Junit. Junit je framework za testiranje u Javi I on se veoma lako generiše. U ovom slučaju testiramo sve klase koje su generisane od strane našeg modela koji smo napravili u PowerDesigneru.

Animacija test

```
package implementation;

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.*;

class AnimacijaTest {
    @Test
    void animiraj() {
        System.out.println("Provera animiranja!");
        String unos = "Uspesno ste animirali!";
        Animacija novaAnimacija = new Animacija();
        String result = novaAnimacija.animiraj();
        assertEquals(unos, result);
    }

    @Test
    void animirajModel() {
        System.out.println("Provera animiranja modela");
        String unos = "Model je uspesno animiran!";
        Animacija novaAnimacija = new Animacija();
        String result = novaAnimacija.animirajModel();
        assertEquals(unos, result);
    }
}
```

Slika 5 Junit kod animacije

U našem slučaju test prolazi i vraća poruku “Uspesno ste animirali!”, što je i očekivano.

Inženjer test

```java
package implementation;

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.*;

class InzenjerTest {

    @Test
    void sacuvaj() {
        System.out.println("Provera cuvanje inzenjera!");
        String unos = "Uspesno ste sacuvali inzenjera!";
        Inzenjer inzenjerTemp = new Inzenjer();
        String result = inzenjerTemp.sacuvaj();
        assertEquals(unos, result);
    }
}
```

Slika 6 Junit kod inženjera

U našem slučaju test prolazi i vraća poruku “Uspesno ste sačuvali inzenejra!”, što je i očekivano.

Jezik test

```java
package implementation;

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.*;

class JezikTest {

    @Test
    void sacuvaj() {
        System.out.println("Provera cuvanja jezika!");
        String unos = "Uspesno ste sacuvali jezik!";
        Jezik noviJezik = new Jezik();
        String result = noviJezik.sacuvaj();
        assertEquals(unos, result);
    }
}
```

Slika 5 Junit kod jezika

U našem slučaju test prolazi i vraća poruku “Uspesno ste sačuvali jezik!”, što je i očekivano.

Model test

```java
package implementation;

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.*;

class ModelTest {

    @Test
    void exists() {
        System.out.println("Provera postojanja modela!");
        boolean provera = true;
        Model modelTemp = new Model();
        boolean result = modelTemp.exists();
        assertEquals(provera,result);
    }

    @Test
    void animiraj() {
        System.out.println("Provera animiranja modela");
        String unos = "Uspesno ste animirali!";
        Model modelTemp = new Model();
        String result = modelTemp.animiraj();
        assertEquals(unos, result);
    }
}
```

Slika 5 Junit kod modela

U našem slučaju testiramo da li je promenljiva provera postavljena na true. Ovaj test prolazi.

Pisanje funkcionalnosti test

```java
package implementation;

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.*;

class PisanjeFunkcionalnostiGUITest {

    @Test
    void sacuvajZapisanuFunkciju() {
        System.out.println("Provera cuvanja zapisane funkcije!");
        String unos = "Uspesno ste sacuvali zapisanu funkciju";
        PisanjeFunkcionalnostiGUI pisanjefunkcionalnosti = new PisanjeFunkcionalnostiGUI();
        String result = pisanjefunkcionalnosti.sacuvajZapisanuFunkciju();
        assertEquals(unos, result);
    }

    @Test
    void odabraoJezik() {
        System.out.println("Provera biranje jezika!");
        boolean odabrao = false;
        PisanjeFunkcionalnostiGUI pisanjefunkcionalnosti = new PisanjeFunkcionalnostiGUI();
        boolean result = pisanjefunkcionalnosti.odabraoJezik();
        assertEquals(odabrao, result);
    }

    @Test
    void biraJezik() {
        System.out.println("Biranje jezika test!");
        String unos = "Izaberite vas programski jezik";
        PisanjeFunkcionalnostiGUI pisanjefunkcionalnosti = new PisanjeFunkcionalnostiGUI();
        String result = pisanjefunkcionalnosti.biraJezik();
        assertEquals(unos, result);
    }
}
```

Slika 5 Junit kod pisanja funkcionalnosti

Pravljenje nivoa test

```java
package implementation;

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.*;

class PravljenjeNivoaGUITest {

    @Test
    void prikaziGui() {
        System.out.println("Prikaz pravljenje nivoa GUI");
        String unos = "Gui se sada prikazuje";
        PravljenjeNivoaGUI pravljenjeNivoaTemp = new PravljenjeNivoaGUI();
        String result = pravljenjeNivoaTemp.prikaziGui();
        assertEquals(unos, result);
    }
}
```

Slika 5 Junit kod pravljenje nivoa

Sistem test

```java
package implementation;

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.*;

class SistemTest {

    @Test
    void prikazGui() {
        System.out.println("Provera prikazivanja GUI!");
        String unos = "Gui se sada prikazuje";
        Sistem sis = Sistem.getsistem();
        String result = sis.prikazGui();
        assertEquals(unos, result);
    }
}
```

Slika 5 Junit kod sistema

Zaključak

Obzirom na to da je sistem već bio dizajniran pre nastanka ove dokumentacije a ponovnom revizijom usled analize za izradu iste primetio sam dosta stvari koje bi mogle da se dodaju ili neke da se promene, ali samim tim bi se trebala ceo sistem promeniti.