Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/16BE45Petrovic

IMPLEMENTACIJA GENERATORA KODA ZA TROSLOJNE POSLOVNE APLIKACIJE

IMPLEMENTATION OF A CODE GENERATOR FOR THREE-TIER BUSINESS APPLICATIONS

Nikolina Petrović, *Fakultet tehničkih nauka, Novi Sad*

Oblast – SOFTVERSKO INŽENJERSTVO I INFORMACIONE TEHNOLOGIJE

Kratak sadržaj – *U radu je opisan razvoj generatora koda za troslojne poslovne aplikacije. Detaljno je opisana specifikacija i implementacija generatora.*

Ključne reči: *generator koda, poslovne aplikacije, crud operacije, model driven development.*

Abstract – *The paper describes the development of a code generator for three-layer business applications. The specification and implementation of the generator is described in detail.*

Keywords: *code generation, business applications, CRUD operations, model driven development.*

1. UVOD

Napredovanje tehnologije, kao i globalna dostupnost kako računara tako i interneta dovela je do širenja tržišta i povećanog broja krajnjih korisnika. Savremeni marketing iziskuje posedovanje aplikacija dostupnih na internetu, koje omogućavaju proširenje biznisa i veći broj kupaca. U online svetu vlada velika konkurencija u svim sferama poslovanja, i s obzirom na količinu sadržaja koji se svakodnevno plasiraju, vrlo je teško i zahtevno biti jedinstven i drugačiji. Kada smislimo aplikaciju koja je jedinstvena, veoma je bitno da na vreme reagujemo.

Takođe, iz godine u godinu na tržištu se javlja sve veći broj firmi koje se bave razvojem softvera. Performanse su samo jedan aspekt ukupnog kvaliteta softvera i obično nisu i najvažniji. Bitan aspekt izrade svake aplikacije predstavlja vreme izrade softvera, kao i cena koja zavisi od vremena uloženog za izradu istog. Postavlja se pitanje kako povećati produktivnost i skratiti vreme izrade softvera.

U ranim fazama izrada aplikacija, često pišemo kod koji se ponavlja. Vreme, koje predstavlja veoma bitan faktor u izradi aplikacija, koristimo za pisanje takozvanih *CRUD* (create, retrieve, update, delete) operacija koje za svaki entitet izgledaju isto. *CRUD* operacije su operacije koje služe za kreiranje, dobavljanje, izmenu i brisanje entiteta iz softverskog rešenja.

Entiteti su u ranim fazama razvoja aplikacije veoma jednostavni, a radi smanjenja grešaka u kasnijem razvoju, kao i lakšeg održavanja softvera, teži se da entiteti takvi i ostanu. Automatizacijom ranih faza izrada softvera možemo drastično smanjiti vreme izrade istog. Takođe, dobro osmišljen i napravljen model koji će se koristiti za automatizaciju ranih faza može dovesti do izrade celog softvera u rekordnom vremenu. Automazitizaciju razvoja softverskih aplikacija možemo postići razvojem automatskih generatora aplikacija ili generatora određenih delova aplikacija (klasa, kontrolera, repozitorijuma..). Takvi generatori će smanjiti količinu rada programera, povećati njegovu produktivnost i smanjiti vreme izrade samog softverskog rešenja.

Tema rada prestavlja razvoj generatora koda koji će omogućiti korisnicima (programerima) automatizovanje ranih faza razvoja softvera i samim tim povećati brzinu izrade programskog rešenja. Generator će omogućiti generisanje *CRUD* operacija i šablona za prikaz podataka. Uz dobro izmodelovan ulaz u generator koda, korisnik će imati mogućnost da bez bilo kakvih izmena dobije potpuno funkcionalno programersko rešenja. .

2. KORIŠĆENE TEHNIKE I TEHNOLOGIJE

Generator koda predstavlja aplikaciju koja služi za automatizovano kreiranje drugih softverskih komponenti. Svaki generator koristi određeni ulaz i na osnovu definisanih pravila generiše izlaz iz generatora.

2.1. Model generatora koda

Generator, koji se još naziva i procesor šablona, koristi dati model podataka ili šablonske stranice, integriše ih na taj način što odgovarajuće podatke modela postavlja u za to označena mesta na šablon stranicama i proizvodi odgovarajuć izlaz.

Generator koda koji će biti implementiran kao ulaz koristi *UML* dijagram. Izlaz iz generatora koda će biti *SpringBoot* [1] modeli, kontroleri i repozitorijumi, kao i *Freemarker* [2] šabloni za grafički prikaz entiteta.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bila prof. dr Gordana Milosavljević.

Inženjerstvo vođeno modelima predstavlja format za pisanje i implementaciju softvera brzo, efikasno i uz minimalne troškove. Ovaj pristup se fokusira na konstrukciju softverskog modela. Model je dijagram koji određuje kako bi softverski sistem trebalo da radi pre generisanja koda.

Paradigma modeliranja smatra se efikasnom ako njeni modeli imaju smisla sa gledišta korisnika koji je upoznat sa domenom i ako mogu poslužiti kao osnova za implementaciju sistema.

Modeli su razvijeni kroz opsežnu komunikaciju između menadžera proizvoda, dizajnera, programera i korisnika domenskih aplikacija. Kako se modeli približavaju završetku, omogućuju razvoj softvera i sistema.

Ovaj princip daje prednosti u produktivnosti u odnosu na druge razvojne metode jer model pojednostavljuje inženjerski proces.

Cela logika je napisana u *UML* dijagramima koji ne zavise od platforme na kojoj će aplikacija biti korištena, te je ovaj način pisanja softvera za višekratnu upotrebu. Takođe, prednost ovakvog pristupa jeste to što omogućava trenutno validaciju korisničkih zahteva i samim tim smanjuje mogućnost greške u ranim fazama. Radikalno smanjuje razvoj, vreme i troškove poslovnih aplikacija.

3. SPECIFIKACIJA

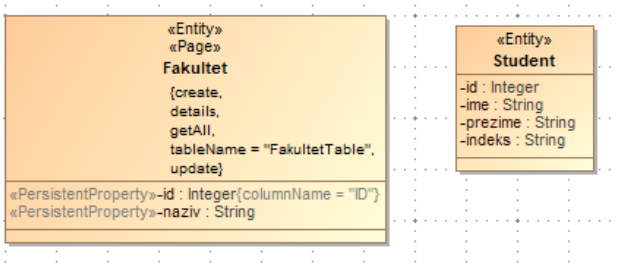
Generator koda koji će biti implementiran predstavlja *CRUD* generator *Java Spring* aplikacije, uz podršku i kreiranje *FreeMarker* šablona. Specifikacija entiteta, kao i međusobne relacije između istih, se zadaje u vidu *UML* dijagrama.

Semantika ovog dijagrama će biti objašnjena u nastavku ovog poglavlja. Takođe, pored specifikacije ulaznog dela generatora, biće specificiran i izlaz generatora.

Slika 3.1. *Model generatora koda*

Ulaz u generator koda, kao što je ranije rečeno, predstavlja *UML* dijagrtam. Kako bi generator koda ispunio svoj maksimalni kapacitet, i količina generisanog koda bila dosta veća u odnosu na količinu koda napisanu od strane programera, posebnu pažnju je potrebno obratiti na prvi korak – kreiranje dijagrama. Dobro struktuirani ulazni podaci će rezultirati kvalitetnim i upotrebljivim rezultatima na izlazu.

Na slici 3.2. prikazan je pojednostavljen ulaz u generator koda. *UML Stereotypes* [3] su mehanizmi koji služe za proširivanje vokabulara *UML*-a, kako bi omogućili kreiranje novih elemenata modela. U praksi, primena odgovarajućih stereotipa modele može učiniti razumljivim i čitljivijim.



Slika 3.2. *Pojednostavljen ulaz u generator koda*

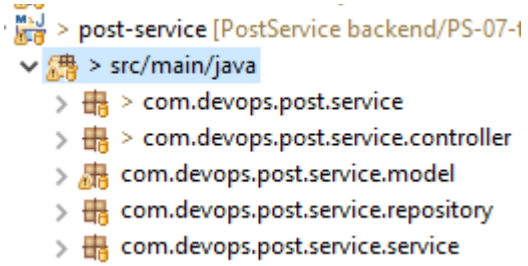
Za potrebe generatora koda opisanog u radu korišćeni su stereotipi *Entity* i *Page*.

Entity stereotip – generatoru koda govori da klasa koja ga sadrži u datom modelu predstavlja entitet. Za dati entitet biće kreirana odgovarajuća *Java* klasa – model.

Page stereotip – generatoru koda govori da je u pitanju klasa koja će predstavljati stranicu u generisanoj aplikaciji. Za klase koje sadrže ovaj stereotip će, pored modela (koji će biti kreiran kod *entity* stereotipa) biti kreirani i odgovarajući kontroler, repozitorijum i servis kako bi se omogućila komunikacija sa bazom podataka. Takođe, biće kreiran i *FreeMarker* šablon za grafički prikaz svih operacija. Očekivani izlaz iz generatora koda predstavlja *Java* izvorni kod i *FreeMarker* šabloni.

Nakon modelovanja *UML* dijagrama koji predstavljaju ulaz u generator koda, korisnik generatora bira putanju na kojoj želi da kod bude generisan, kao i naziv paketa u kom želi da klase budu generisane.

Na ovaj način, korisnik ima potpunu slobodu i generator koda može da koristi u već postojećim projektima, kao i u već implementiranim paketima. Kako bi se generisane klase razlikovale od postojećih, dodat je sufiks *Gen* (skraćeno od generisan), kao i propratni komentar u svakoj od klasa.



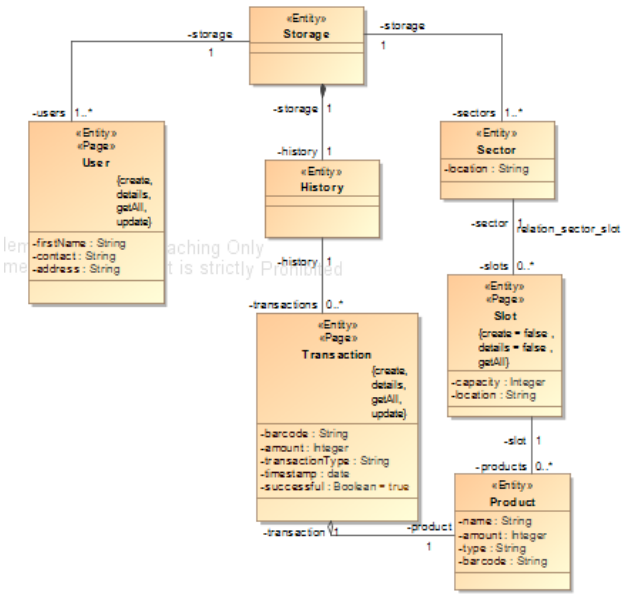
Slika 3.2. Prikaz strukture paketa

4. IMPLEMENTACIJA

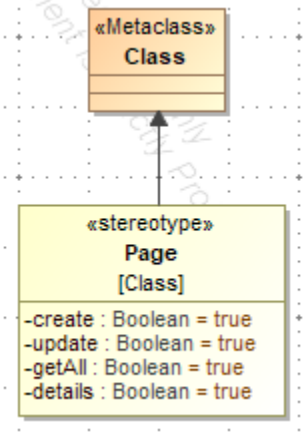
Ulaz u generator koda predstavlja UML dijagram. Na slici 4.1. prikazan je potpuno funkcionalan dijagram, koji je korišten kao primer za potrebe generatora, a čija će primena biti detaljnije opisana u narednim poglavljima.

Za opis klasa su korišteni *UML* stereotipi. Kao što je ranije rečeno, glavna dva tipa stereotipa neophonda za izradu i razumevanje generatora su *Entity* i *Page*. Sami stereotipi predstavljaju proširenje postojeće metaklase.

Stereotip *Page* nam govori da klasa na kojoj se on nalazi treba da bude stranica, odnosno da je neophodno kreirati odgovarajući repozitorijum, servis i kontroler, kao i šablone.



Slika 4.1. Primer ulaza u generator koda



Slika 4.2. Stereotip Page

Na slici 4.2. prikazan je stereotip Page, sa svojim označenim vrednostima. Objašnjenje označenih vrednosti:

1. *create* – ukoliko je vrednost *true*, kreiraće se sve neophodne metode za kreiranje entiteta opisanog datom klasom
2. *update* – ukoliko je vrednost taga *true*, kreiraće se sve neophodne metode za izmenu entiteta opisanog klasom
3. *getAll* – ukoliko je vrednost tačna, kreiraće se sve neophone metode za vraćanje liste svih objekata tipa datog entiteta
4. *details* – ukoliko je vrednost *true* biće kreirana

Generator koda generiše kontrolere, servise, modele i repozitorijume koji su definisani ulazom u generator. Kreiraju se neophodne klase, kao i neophodne zavisnosti među istim.
Metode koje su generisane unutar kontrolera i servisa su sledeće:

1. *findById* – pronalazak entiteta pomoću jedinstvenog identifikatora
2. *add* – dodavanje novog entiteta
3. *update* – izmena postojećeg entiteta
4. *delete* – brisanje entiteta
5. *findAll* – pronalazak svih entiteta

.
Takođe, generator koda kreira *FreeMarker* šablone koji će služiti za grafički prikaz aplikacije. U zavisnoti od vrednosti stereotipa, biće kreirani šabloni za dodavanje, prikaz svih, detaljni prikaz jednog entiteta, kao i izmenu entiteta sa unapred popunjenim podacima entiteta kojeg želimo da izmenimo.

5. PRIMENA GENERATORA KODA

U poglavlju koje sledi biće opisana dva projekta u čijim procesima implementacije je iskorišten generator koda. Statistički prikaz će pokazati koja količina koda je generisana i koliko je proces izrade ubrzan.

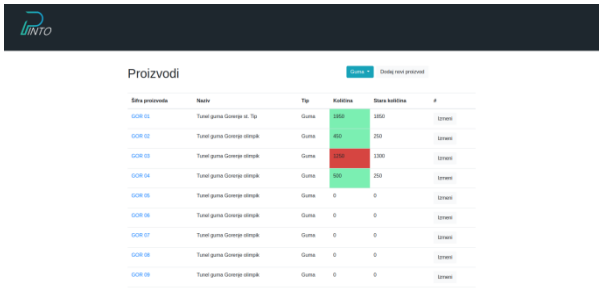
Aplikacija za skladište - U specifikaciji aplikacije dato je da je neophodno kreirati skladište za proizvode. U pitanju je firma čiji se asortiman deli na proizvode od plastike i proizvode od gume. Od operacija bilo je potrebno kreirati dodavanje novog proizvoda, prikaz svih proizvoda (kao i sortiranje po tipu), izmenu proizvoda (akcenat je stavljen na količinu) i prikaz detalja odabranog proizvoda. Zahtev korisnika je da u tabeli, ukoliko je prethodna količina bila manja od nove unete količine to polje bude crvene boje, kako bi mu signaliziralo da je potrebno da dopuni skladište.

Ukoliko su dodati novi proizvodi, te je nova količina veća od prethodne, potrebno je polje obojiti u zelenu boju. Takođe, na zahtev korisnika u tabeli je dodato polje Izmeni koje ga direktno vodi do stranice za izmenu proizvoda kako bi se proces izmene dodatno ubrzao.

	Model	Repozitorijum	Servis	Kontroler
Broj ručno pisanih metoda	0	1	1	1
Broj generisanih metoda	0	0	3	4
Broj generisanih linija koda (ukupno za sve entitete)	44	14	29	54
Broj ručno pisanih linija koda (ukupno za sve entitete)	0	1	3	4
Procenat generisanog koda	100%	93%	90%	93%

Slika 5.1. Statistički prikaz generisanog koda

Na slici 5.2. prikazan je izgled aplikacije za čiju implementaciju je korišten generator koda.



Slika 5.2. Prikaz aplikacije za skladište

Platforma za deljenje fotografija na internetu - Tema zadatka jeste bila razvijanje platforme (društvene mreže) za deljenje fotografija na internetu. Arhitekturu projekta je bilo potrebno razviti u vidu mikroservisa.
Mikroservisna arhitektura je princip razvoja aplikacija u obliku malih, izdvojenih i nezavisnih servisa koji komuniciraju putem jednostavnih mehanizama kao što je HTTP API.

	Model	Repozitorijum	Servis	Kontroler
Broj ručno pisanih metoda	0	2	4	4
Broj generisanih metoda	0	0	3	4
Broj generisanih linija koda (ukupno za sve entitete)	124	13	31	54
Broj ručno pisanih linija koda (ukupno za sve entitete)	0	2	37	18
Procenat generisanog koda	100%	86%	46%	75%

Slika 5.3. Statistički prikaz generisanog koda *Post* servisa

	Model	Repozitorijum	Servis	Kontroler
Broj ručno pisanih metoda	0	2	4	8
Broj generisanih metoda	0	0	3	4
Broj generisanih linija koda (ukupno za sve entitete)	75	13	31	54
Broj ručno pisanih linija koda (ukupno za sve entitete)	0	2	76	53
Procenat generisanog koda	100%	86%	29%	50.4%

Slika 5.4. Statistički prikaz generisanog koda *Account* servisa

	Model	Repozitorijum	Servis	Kontroler
Broj ručno pisanih metoda	0	6	5	5
Broj generisanih metoda	0	0	6	8
Broj generisanih linija koda (ukupno za sve entitete)	122	27	58	108
Broj ručno pisanih linija koda (ukupno za sve entitete)	0	6	51	28
Procenat generisanog koda	100%	81%	53%	73%

Slika 5.5. Statistički prikaz generisanog koda *Account* servisa

Campaign service, *Post service* i *Account service* servisi predstavljaju ključni deo aplikacije i deo aplikacije na kome je korišten generator koda.

Kako je komunikacija između servisa bila napravljena u vidu mikroservisa, relacije između entiteta bile su ručno čuvane u vidu jedinstvenih identifikatora i bile pozivane kada je to bilo neophodno.

6. ZAKLJUČAK

Osnovni motiv za razvoj generatora koda predstavlja povećavanje produktivnosti vremena utrošenog od strane programera za izradu projekta automatizacijom ranih faza izrade. Automatizacija svakodnevnih zadataka i poslova u velikoj meri može povećati efikasnost kako pojedinca, tako i firme. Generator koda opisan u radu u velikoj meri automatizuje rane faze izrade svakog web rešenja napisanog u pomenutim tehnologijama. Pored toga, zbog ulaza u sistem koji predstavlja UML dijagram programeru omogućava vizualni pregled klasa i lakše uviđanje potencijalnih grešaka ili mogućih poboljšanja samog modela.
Kreiranjem modela, kontrolera, servisa, repozitorijuma i FreeMarker šablona, generator koda nakon generisanja programeru daje potpuno funkcionalno programsko rešenje.
Generator koda je moguće proširiti dodavanjem metoda koje se, pored CRUD operacija, veoma često koriste u programskim rešenjima – metode za filtriranje i sortiranje sadržaja. Idealno rešenje bi bilo da se korisniku omogući odabir metoda koje su mu neophodne, kao što je urađeno za CRUD operacije pomoću označenih vrednosti. Pored metoda za filtriranje i sortiranje, generator bi bilo poželjno proširiti i metodama za pretragu po određenim parametrima ili boolean vrednostima (prikaži sve aktivne korisnike,..).

7. LITERATURA

[1] *Spring* Boot, https://spring.io/projects/spring-framework, preuzeto septembra 2021.
[2] *FreeMarker*, https://freemarker.apache.org//, preuzeto septembra 2021.
[3] UML Stereotypes, https://www.visual-paradigm.com/tutorials/how-to-create-stereotyped-model-element.jsp, preuzeto septembra 2021

Kratka biografija:

Nikolina Petrović rođena je 03. marta 1997. godine u Beogradu. U Inđiji je završila osnovnu školu „Dušan Jerković“. Nakon toga upisuje računarsku gimnaziju „Smart“ u Novom Sadu i završava je 2016. godine. Upisuje Fakultet tehničkih nauka, smer Softversko inženjerstvo i informacione tehnologije. Studije završava u roku, 2020. godine. Iste godine upisuje master akademske studije, smer Softversko inženjerstvo i informacione tehnologije.