Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/03BE10Vojnovic

RAZVOJNO OKRUŽENJE ZA GENERISANJE PROGRAMSKOG KODA SPRING API REST APLIKACIJA

DEVELOPMENT ENVIRONMENT FOR GENERATING SPRING API REST APPLICATIONS

Jana Vojnović, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U ovom radu predstavljeno je softversko rješenje za generisanje koda Spring API REST aplikacija. Pomenuto rješenje preuzima metapodatke iz šeme baze podataka i u kombinaciji sa unaprijed definisanim šablonima generiše izvršiv programski kod.*

Ključne reči: *Generator koda, šablon, metapodaci*

Abstract – *This paper presents a software solution in favor of code generation for Spring API REST applications. The solutions uses predefined templates and metadata extracted from the database to generate runnable programming code.*

Key words: *code generator, template, metadata*

1. UVOD

U svijetu informacionioh tehnologija koje prate agilne metodologije razvoja softvera [1], sve više se javlja potreba da se proizvod, u ovom slučaju programski kod koji pruža određenu funkcionalnost, što brže i efikasnije plasira na tržište. U cilju podizanja kvaliteta koda na što viši nivo, kao i brzine kojom se dostavlja isti, pojavile su se tehnike automatskog generisanja programskog koda. Primenom pomenutih tehnika, softverski inženjer piše kod na višem nivou apstrakcije koji služi kao strukturalni šablon i osnova za automatizaciju procesa pisanja programskog koda. Ovaj pristup je usko vezan sa inženjerstvom vođenim modelima (engl. MDE) [1], koji podrazumjeva da je prvi korak zapravo kreiranje modela.

Kao jedan od pravaca u kome ide MDE jeste i generisanje koda na osnovu metapodataka šeme baze podataka (BP). U ovom slučaju, struktura šeme BP se koristi kao jedinstven izvor informacija.

Savremeni softverski sistemi najčešće primjenjuju objektno-relacione mapere (eng. ORM) da bi povezali torke iz tabela relacione šeme BP u klase objektne reprezentacije aplikativnog sloja. Uzevši u obzir realne potrebe korisnika, u ovom slučaju softverskog inženjera, osnovna primjena ovog softverskog rješenja jeste podrška u implementaciji procesa objektno relacionog mapiranja. U radu će biti opisano projektovanje i implementacija komponente generatora koda čija je svrha automatizacija repetitivnih procesa prilikom razvoja softvera, a koje se tiču pisanja programskog koda.

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

2. KORIŠĆENE TEHNOLOGIJE

Projektno rješenje u ovom radu predstavlja softverski alat JGen napisan u programskom jeziku Java [3], gdje je Spring Framework [4] odabran kao radni okvir. Dio softverskog rješenja zaduženog za generisanje programskog koda će pristupati šemi BP na osnovu zadate konfiguracije i iz nje dobaviti metapodatke. Za pomenuti proces su kao izvor podataka odabrane relacione BP. Glavni razlog jeste njihova rasprostranjenost u poslovnim sistemima, ali i jasna struktuiranost i uniformnost kroz cijelu šemu BP. Kao sistem za upravljanje bazama podazaka odabran je MySQL [5].

Pomenuti metapodaci će uz pomoć šablona napisanog u FreeMarker [6] jeziku, biti osnova za proces generisanja programskog koda koji obuhvata:

- Model klase, koje služe kao reprezentacija tabela iz šeme BP. Ove klase će koristiti Hibernate [7] ORM tako da će sam proces mapiranja torki iz šeme BP biti u potpunosti kompatibilno sa poljima iz klasa;
- JPA repozitorijum, koji služi kao sloj komunikacije sa šemom BP. Za implementaciju ovog sloja je referenciran Spring Data [8] repozitorijum;
- Servis, koji služi kao međusloj između repozitorijuma i kontrolera. U ovom dijelu će biti implementirane osnovne operacije manipulacije nad podacima;
- Kontroler, koji predstavlja krajnji sloj u ovoj aplikaciji. S obzirom da je aplikacija generisana na osnovu MVC dizajn šablona [9], ovaj kontroler će biti implementiran kao REST kontroler;

Izgenerisana aplikacija se oslanja na pomenute tehnologije i alate korištene prilikom implementacije softverskog rješenja zaduženog za njeno generisanje. Treba uočiti da jezik u kome je napisan sam softver za generisanje koda ne mora biti isti kao i izgenerisani kod, u opštem slučaju. Takođe, u izgenerisanoj aplikaciji može se koristiti i bilo koja relaciona baza, pod uslovom da postoji kompatibilna biblioteka za Hibernate ORM koji je impementiran na aplikativnom nivou.

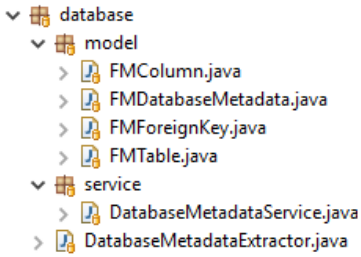
3. IMPLEMENTACIJA RJEŠENJA

Rješenje opisano u ovom radu se sastoji iz više modula koji ujedno predstavljaju i logičke cjeline. Pomenuti moduli, kao i načini njihove implementacije i zaduženja u

procesu generisanja, će biti detaljnije opisani u narednim poglavljima.

3.1. Modul za dobavljanje metapodataka

Modul za dobavljanje metapodataka se nalazi u paketu *database* (Slika 1) i zadužen je da ostvari konekciju na zadatu šemu BP i dobavi informacije iz nje.



Slika 1. *Paket modula za dobavljanje metapdoataka*

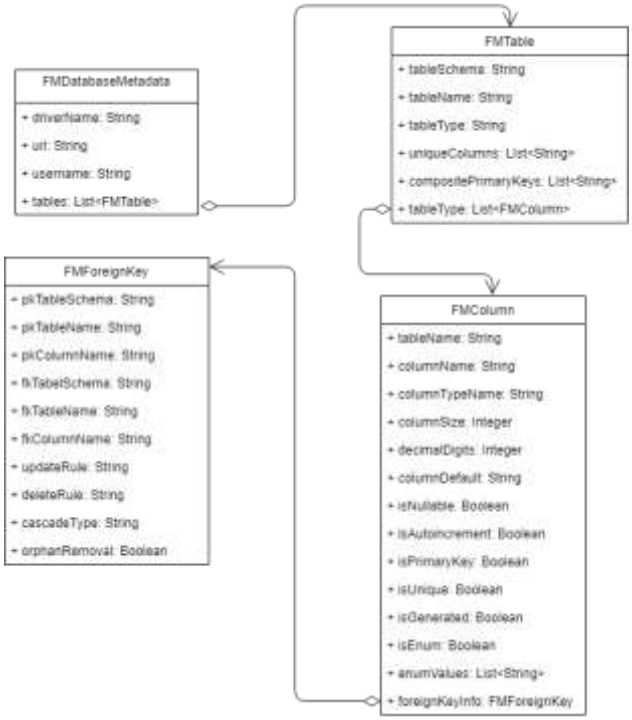
Klase definisane u ovom modulu (slika 2) predstavljaju reprezentaciju relacionih metapodataka u ovom projektu. Za dobavljanje svih metapodataka iz šeme BP korištena je JDBC konekcija. Direktno iz pomenute konekcije se poziva metoda *getMetaData* koja dobavlja metapodatke u obliku klase *DatabaseMetadata* [10]. Metode koje su korištene su: metoda za dobavljanje podataka o tabelama, metoda za dobavljanje podataka o kolonama u tabeli, metoda za dobavljanje podataka o stranim ključevima, metoda za dobavljanje podataka o indeksima unutar tabele i metoda za dobavljanje podataka o primarnim ključevima u tabeli.

Kao rješenje u većini savremenih softverskih sistema jeste ograničavanje korisnika na aplikativnom nivou o unosu enumeracije, odnosno nabrojivih vrijednosti. U tom slučaju su ova polja u šemi BP zapravo obična tekstualna polja, a samo na aplikativnom nivou imaju predefinisan skup mogućih vrijednosti. Prilikom izrade ovog rješenja posvećena je posebna pažnja na dobavljanje predefinisanih enum vrijednosti iz šeme BP i kreiranje klase koja ih oslikava na nivou objekata u reprezentaciji Java programskog jezika.

Prilikom preuzimanja svih kolona koje pripadaju referentnoj tabeli koristi se metoda *getColumns* pomenute klase *DatabaseMetadata*. Tokom pomenutog procesa nije moguće razlikovati kolone primarnog ključa od običnih kolona samo na osnovu skupa informacija koje ova metoda dobavlja. Iz ovog razloga je korištena metoda posebno za tu namjenu pod nazivom *getPrimaryKeys*. Pomenuta metoda dobavlja sve kolone primarnog ključa iz prosleđene tabele.

Proces dobavljanja informacija o primarnom ključu u ovom projektu je realizovan tako da se prvo dobave sve kolone referentne tabele. Zatim se dobave sve kolone primarnog ključa iste tabele. Pomenute kolone se zatim uporede i ukoliko se poklapaju, zaključak je da referencirana kolona spada pod ograničenje primarnog ključa.

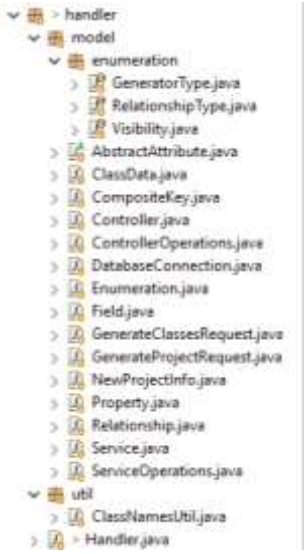
Ukoliko tabela ima više od jedne kolone pod ograničenjem primarnog ključa, to znači da ona ima kompozitni primarni ključ. Pomenute informacije se tretiraju na poseban način prilikom procesa generisanja aplikacije. Na slici 2 je prikazan dijagram klasa za modul za dobavljanje metapodataka.



Slika 2. *Dijagram klasa za database modul*

3.2. Modul za transformaciju podataka

Modul za transformaciju metapodataka se nalazi u paketu *handler* (Slika 3) i zadužen je da podatke dobijene iz modula za dobavljanje metapodataka transformiše u oblik koji će više odgovarati objektnoj reprezentaciji. U paketu *model* se nalaze model klase koje po svojoj strukturi predstavljaju metamodel ovog softverskog rješenja.



Slika 3. *Paket modula za transformaciju podataka*

Klasa *Handler* je komponentna klasa ovog modula u kome se obavljaju sve transformacije. Prilikom prosleđivanja podataka za generisanje kontroleru koji je povezan sa pomenutom *Handler* klasom, moguće je specificirati dvije vrste generisanja:

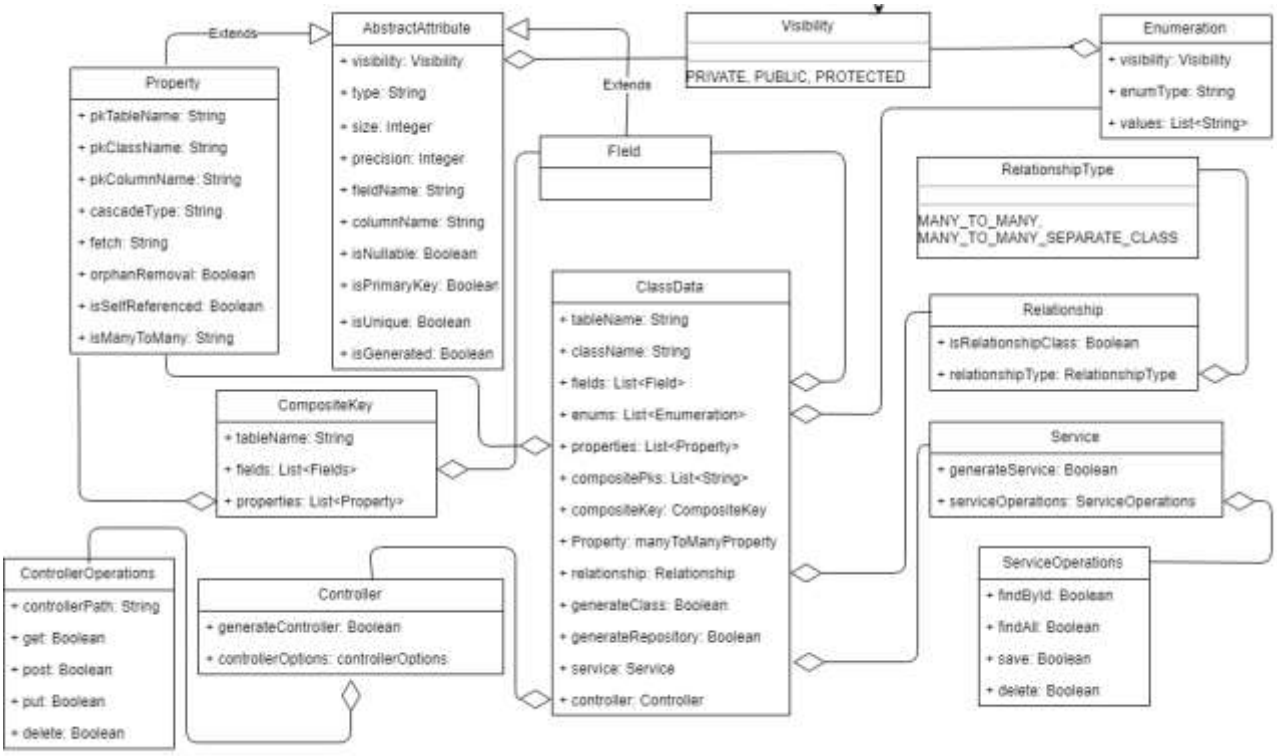
- Potpuno novi projekat - u ovom slučaju će će biti generisani svi potrebni elementi za novi Spring Boot Maven projekat, koji podrazumjevaju dodatne klase i konfiguracione datoteke;

- Integracija u postojeći projekat – u ovom slučaju će biti generisane samo odabrane klase u MVC dizajn šablonu, tj. model, repozitorijum, servis i kontroler klase u već postojećem Spring projektu.

Na slici 4 prikazan je dijagram klasa za modul za transformaciju podataka.

- *CompositeKey* klase, koje objedinjuju sva polja ograničenja primarnog ključa, ukoliko ono obuhvata više od jedne kolone;

Prilikom implementacija ovog rješenja poseban osvrt je na mapiranju kompozitnih, tj. složenih primarnih ključeva iz relacione u objektnu reprezentaciju. U tom procesu korišćen je javax.persistence interfejs na koji se Hibernate maper oslanja. Važno je napomenuti da je podržan i slučaj ukoliko je strani ključ dio primarnog ključa.



Slika 4. *Dijagram klasa za handler modul*

Shodno odabranom tipu generatora razdvojena je i logika unutar samog softverskog rješenja. Svaka tabela preuzeta u modulu za dobavljanje metapodataka u obliku klase *FMTable* se u ovom modulu transformiše u klasu *ClassData*. Slučaj kada tabela iz šeme BP neće biti prevedena u klasu objektne reprezentacije jeste ukoliko je to poveznička tabela sa samo dva strana ključa. Prilikom realizacije same *Handler* klase, ona je izdjeljena u više manjih logičkih cjelina koje su ostvarene kroz privatne metode prateći dizajn šablon razdvajanja odgovornosti.

Nakon incijalnog kreiranja objekata tipa *ClassData*, obrađuju se i kolone koje će postati njena polja. Sama hijerahija kolona podrazumjeva da je *AbstractAttribute* apstraktna klasa koja sadrži zajedničke informacije za bilo koji tip polja. Zatim se podjela pomenutih polja vrši na:

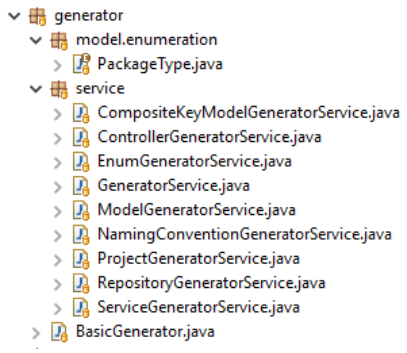
- *Field* klase, koje predstavljaju obične kolone iz šeme BP uključujući i kolone primarni ključ, ukoliko je ovo ograničenje samo nad jednom kolonom;
- *Property* klase, koje predstavljaju referencijalne integritete iz šeme BP;
- *Enumeration* klase, koje predstavljaju kolone tipa nabrajanja;

U relacionoj reprezentaciji modela tabele mogu biti povezane vezom više ka više, pri čemu se kreira poveznička tabela koja u sebi sadrži referencijalne integritete povezanih tabela. U objektnoj reprezentaciji poveznička tabela se, većinom, ne mapira na povezničku klasu, osim ukoliko korisnik to eksplicitno ne želi. Maperi kao što je Hibernate vrlo jasno prepoznaju pomenuti tip veze, pa će se povezničke tabele biti prepoznate i po potrebi generisane na strani šeme BP.

Prilikom implementacije ovog rješenja pojavio se problem prepoznavanja povezničkih tabela, kao i njihova prezentacija u objektni domen. Pored tabela koja su jasno povezničke i sadrže samo dvije kolone koje predstavljaju referencijalne integritete povezničkih tabela, javljaju se i tabele koje sadrže dodatne kolone. Ovakve pomenute tabele bi trebalo mapirati kao posebne klase, ukoliko želimo da očuvamo postojanost pomenutih kolona.

3.2. Modul za generisanje programskog koda

Modul za generisanje pogramskog koda se nalazi u paketu *generator* (Slika 5) i zadužen je da iz ulaznih podataka generiše izvršivi programski kod.



Slika 5. *Paket modula za generisanje koda*

Modul za generisanje koristi model klase iz *handler* paketa uz napomenu na implementirane validacije na poljima i implementaciju posebnog validatora kao u cilju sprečavanja korisnika da unese nevalidne podatke. Uzimajući Java programski jezik za odredišni kod, javlja se potreba za importima (uvozima) u svakoj izgenerisanoj klasi. Njihovo rukovanje je odrađeno dinamički, tako da se neće desiti da neka klasa ima uvoze koje ne koristi. Na taj način se izbjegavaju upozorenja o nekorištenim uvozima. Opisana logika je implementirana u svakoj servis klasi na osnovu sadržaja prosleđenog generatoru.

Ovaj model je ostvaren preko više geneatora definisanih kao servisi koji obavljaju određeni dio logike za generisanje. Svaki generator servis koristi sopstveni šablon. BasicGenerator predstavlja apstraktni generator koga nasleđuju svi ostali generatori. Svaki učitani šablon se mapira na objekat klase Template iz paketa freemarker, korištenog tokom projektovanja.

Prilikom izrade ovog softverskog rješenja pojavio se problem imenovanja tabela prilikom prevođenja iz objektnih klasa u relacionih oblik tabela. Za ovu konvenciju imenovanja odgovoran je maper Hibernate. Naime, od verzije 5 Hibernate-a koristi se kao podrazumjevana konvencija po kojoj prevodi sva imena tabela u oblik sa karakterom „_“. Tako će tabela koja se zove npr. customerDetails biti automatski prevedena u naziv customer_details. S obzirom da je ovo podešavanje neželjeno za ciljnu aplikaciju, bilo je potrebno na neki način pružiti korisniku mogućnost reimplementacije metoda koje sadrži klasa *PhysicalNaming-StrategyStandardImpl* iz paketa org.hibernate.boot.

4. ZAKLJUČAK

U ovom radu predstavljeno je projektno rješenje sistema za generisanja programskog koda namjenjeno za Java Spring API aplikacije. Softversko rješenje predstavlja aplikaciju namjenjenu softverskim inženjerima kao domenskim stručnjacima omogućavajući im brzo projektovanje samostalne aplikacije. Metpodaci iz šeme BP se preuzimaju i prebacuju u tekstualni oblik JSON sintakse, koji se zatim transformiše u jezik specifičan za domen ovog softverskog rješenja. Korisniku je omogućena izmjena podataka u pomenutom obliku kao i uticaj na sam proces generisanja, uz kontrolu i sprečavanje unosa grešaka ili nevalidnih podataka. Od primljenih podataka projektno rješenje zatim generiše Java Spring API aplikaciju sa model klasama koje odgovaraju tabelama iz šeme BP.

Razmatrani pravci daljeg razvoja su unapređenje generatora tako da ih je moguće nezavisno koristiti. Primjer za to je generisanje samo model klasa ili generisanje samo repozitorijuma ili samo servis sloja. Zatim, mogućnost uticaja korisnika na zavisnosti koje se generišu u *pom.xml* datoteku, kao i mogućnost odabira njihovih verzija.

U pogledu proširenja samog softverskog rješenja, kao sledeći korak u razvoju je projektovanje grafičkog interfejsa koji bi komunicirao za postojećim rješenjem. Na taj način bi korisnik imao pregledniji uvid u podatke, za razliku od sadašnjeg tekstualnog fajla. Kao predlog za ovo je web aplikacija napisana u odabranom programskom jeziku koja komunicira preko REST HTTP protokola. Sa druge strane, može se raditi na proširenju generatora za druge jezike i platforme. U tom slučaju se modul za preuzimanje metapodataka može u potpunosti iskoristiti.

5. LITERATURA

[1] G. Milosavljević, materijali sa predmeta „Metodologije brzog razvoja softvera“, Fakultet tehničkih nauka, Novi Sad, 2017

[2] „Model Driven Enigneering“ [Na mreži]. Dostupno na: https://en.wikipedia.org/wiki/Model_driven_engineering . [Pristupljeno: 15-Okt-2018].

[3] „Java Programming Language“ [Na mreži]. Dostupno na: https://docs.oracle.com/javase/8/docs/ [Pristupljeno: 10-Jun-2018].

[4] „Spring Framework“ [Na mreži]. Dostupno na: https://docs.spring.io/spring/docs/current/spring-framework-reference/ [Pristupljeno: 7-Jul-2018].

[5] „MySQL“ [Na mreži]. Dostupno na: https://www.mysql.com/ [Pristupljeno: 17-Okt-2018].

[6] „Apache FreeMarker“ [Na mreži]. Dostupno na: https://freemarker.apache.org/ [Pristupljeno: 4-Okt-2018].

[7] „Hibernate Mapper“ [Na mreži]. Dostupno na: http://hibernate.org/ [Pristupljeno: 22-Okt-2018].

[8] „Spring Data“ [Na mreži]. Dostupno na: https://docs.spring.io/spring-data/jpa/docs/current/reference/html/ [Pristupljeno: 20-Sep-2018].

[9] „Model View Controller Design Pattern“ [Na mreži]. Dostupno na: https://en.wikipedia.org/wiki/Model-view-controller [Pristupljeno: 17-Sep-2018].

[10] „DatabaseMetadata“ [Na mreži]. Dostupno na: https://docs.oracle.com/javase/7/docs/api/java/sql/DatabaseMetaData.html [Pristupljeno: 17-Sep-2018].

Kratka biografija:

Jana Vojnović rođena je na Ilidži 1994. Osnovne studije iz oblasti Računarstvo i automatika – Primjenjene računarske nauke i informatika je završila 2016. godine kada je upisala i master studije iz iste oblasti.