Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/09BE26Borovcanin

GENERATOR KLIJENTSKE I SERVERSKE WEB APLIKACIJE BAZIRAN NA GO PROGRAMSKOM JEZIKU I TOML SPECIFIKACIJI

CODE GENERATOR FOR CLIENT AND SERVER WEB APPLICATION BASED ON GO PROGRAMMING LANGUAGE AND TOML SPECIFICATION

Dušan Borovčanin, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U ovom radu je opisan dizajn domenskog jezika za specifikaciju klijentske i serverske web aplikacije. Projektovan je generator koda koji taj jezik koristi i opisana je implementacija generatora. Generator je implementiran upotrebom obrađivača šablona, a podržana je i integracija sa ručno pisanim kodom.*
Ključne reči: *Generator koda, TOML, šabloni*
Abstract – *This paper presents the DSL for client and server application modeling. The code generator based on the DSL is specified and its implementation is described. Code generator is implemented using template engine. Integration of generated and hand-written code is supported.*

Key words: *Code generator, TOML, templates*

1. UVOD

Efikasnost, u tehničkom smislu, predstavlja odnos ukupnog i korisnog rada. U opštijem smislu, efikasnost se mjeri potrošnjom resursa i kvalitetom finalnog rezultata. Razvoj hardvera i, prije svega, infrastrukture za pristup internetu, te njihova globalna dostupnost, rezultovali su širenjem tržišta i povećanjem broja potencijalnih korisnika. U takvim okolnostima, brzina kojom softver prelazi iz faze ideje u proizvod spreman za upotrebu dobija veliku važnost i često predstavlja ključan faktor za uspjeh na tržištu. Ekspanzija web aplikacija dovela je do učestalog pisanja koda koji se u najvećem broju slučajeva bavi tzv. CRUD (engl. *create, read, update, delete*) operacijama, tj. kreiranjem, dobavljanjem, izmjenom i brisanjem resursa kojim se modeluju entiteti iz nekog realnog sistema. Povećana potreba za ovom vrstom aplikacija je dovela do pisanja koda koji se ponavlja. Sa tehničke strane, povećava se vjerovatnoća greške, otežava održavanje i proces razvoja čini monotonim i bespotrebno troše resursi.

Kao odgovor na ove probleme nastaju tehnike za razvoj softvera vođenog modelima (*model-driven software development*). Generisanje koda utiče na uštedu vremena, i donosi prednosti kao što su uniformnost, minimizacija koda koji se ponavlja i ekspresivnost.

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

Tema ovog rada jeste generator koda koji na osnovu specifikacije zadate u TOML [1] (*Tom's Obvious, Minimal Language*) formatu generiše serversku i klijentsku stranu web aplikacije. Za generator koji je predmet ovog rada u nastavku će se koristiti naziv TCG (*TOML Code Generator*).

Velika popularnost web aplikacija koje počivaju na REST [2] arhitekturi uticala je na nastanak alata i specifikacija za opisivanje API-ja. Iako ove specifikacije mogu biti korištene kao ulaz za generisanje koda, to nije njihova primarna namjena. One pružaju veliku fleksibilnost za detaljno i nedvosmisleno opisivanje API-ja web aplikacije. Upravo ta preciznost i detaljnost rezultira kompleksnom, obimnom i često nedovoljno čitljivom specifikacijom, pa su potrebni dodatni alati kako bi se korisniku olakšao pregled i upotreba. Takođe, po svojoj prirodi se ulazne tačke (engl. *endpoint*) za različite REST operacije razlikuju, pa je potrebno opisati svaku ulaznu tačku za svaku operaciju. Za generator koda REST aplikacije ove informacije mogu se smatrati nebitnim i dovoljno je opisati kako je entitet interno predstavljen u sistemu, u kakvom je odnosu sa drugim entitetima, kao i koje su operacije nad njim dozvoljene.

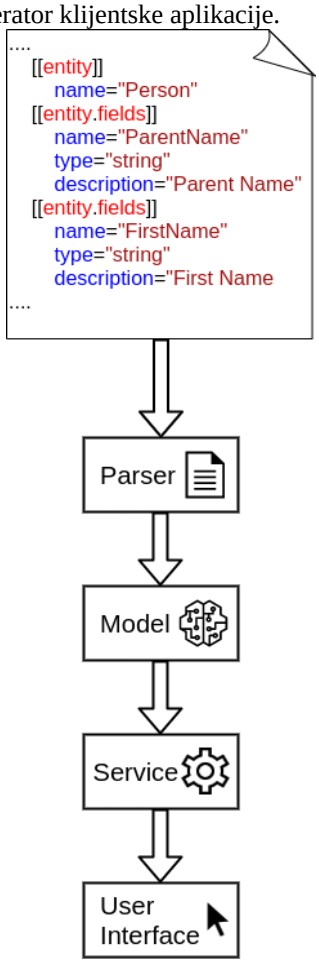
Namjena ovog generatora koda jeste prije svega podrška za implementaciju osnovnih CRUD operacija, pa je bitno da se entiteti i dozvoljene operacije nad njima prikažu na jednostavan i čitljiv način. Za ovu svrhu se može upotrijebiti dobro poznati koncept ključ-vrijednost. Upravo iz tih razloga je za format ulaznog dokumenta odabran TOML. Ovaj format je odabran zbog svoje jednostavnosti i čitljivosti. Za razliku od YAML [3] formata, kome je sličan, indentacija ne utiče na semantiku, što kod čini manje dvosmislenim. Odabran je podskup TOML specifikacije i kreiran mini-jezik sa jednostavnom semantikom koji se koristi kao ulaz u generator koda.

2. SPECIFIKACIJA SISTEMA

TCG je generator serverske i kijentske aplikacije koje pružaju osnovne operacije kreiranja, dobavljanja, izmjene i brisanja resursa. Specifikacija resursa se zadaje u TOML formatu, sa sematikom koja je objašnjena u nastavku. TCG generiše i serversku i klijentsku aplikaciju, pružajući mogućnost brzog prototipskog razvoja. Neki od problema kojima se ovaj generator bavi su: izmjene specifikacije entiteta, kao i integracija sa ručno pisanim kodom ukoliko se te izmjene ne mogu automatizovati.

TCG generator se sastoji iz nekoliko cjelina, pri čemu svakoj cjelini odgovara određena faza u generisanju koda. Na slici 1 su prikazani osnovne komponente TCG generatora:

- Parser ulaznog TCG DSL dokumenta
- Komponenta za generisanje modela
- Generator serverske aplikacije
- Generator klijentske aplikacije.



Slika 1. *Komponente TCG generatora*

Parser TOML fajlova predstavlja ulaznu komponentu u TCG generator. Ulaz u ovu komponentu je specifikacija napisana u TCG DSL jeziku koji koristi pomenutu TOML sintaksu, a izlaz je model u programskom jeziku Go. Parser čita TOML fajl i verifikuje da li je dokument ispravno formiran, mapirajući odgovarajuća polja iz dokumenta na model. Iako je izlaz iz parsera model, parser ne vrši semantičke provjere, već se te provjere izvršavaju u komponenti za generisanje modela. Ukoliko fajl nije u ispravnom formatu, ova komponenta će prijaviti grešku. TOML je, prema zvaničnoj specifikaciji, format za konfiguracione fajlove čiji je cilj da bude jednostavan za čitanje i pisanje, kao i procesiranje u hash mapu, jednu od najčešće korištenih struktura podataka. Osnovna struktura predstavljena je parovima ključ-vrijednost. Ključevi mogu biti prosti (ASCII karakteri i brojevi, crtica i donja crta), složeni (string između znakova “ ili ‘) ili kompleksni (jedan ključ sadrži više hijerarhijskih nivoa razdvojenih tačkom, npr. *entity.field.type=”float”*). Podržani su i nizovi. Jedna od najčešće korištenih osobina TOML jezika su tabele koje predstavljaju heš (engl. hash) tabele/rječnike. One su kontejneri za druge parove ključ-vrijednost. Za imena tabela važe ista pravila kao i za imenovanje ključeva, uključujući i kompleksne ključeve - na taj način se formiraju pod-tabele.

Tri osnovna pojma za opisivanje modela u TCG DSL-u su servis, entitet i polje. Servis je korijenski element koji sadrži jedan ili više entiteta. Entitet sadrži polja, koja predstavljaju opise atributa izgenerisanog koda, uključujući naziv, tip, polje i meta-podatke za rješavanje odnosa u relacionom modelu. Tip može biti prost ili kompleksan. Ukoliko neki entitet ima polja kompleksnog tipa, potrebno je naznačiti u kom su odnosu taj entitet i entiteti koji su tip tog polja. Odnosi mogu biti: *one-to-one*, *one-to-many* i *many-to-one*.

Generisanje modela je semantička provjera modela koji je izlaz iz parsera i uključuje:

- rješavanje zavisnosti koje su u *import* sekciji
- validaciju modela
- rješavanje međusobnih odnosa među enititetima.

Na listingu 1 dat je primjer jednog TOML fajla, koji je ujedno i validan ulaz za TCG generator, koji pokazuje kako izgleda model u TCG DSL-u.

```
name="example"
package="service"
description="""
# This is a nice place to put your short README file.
.
"""
[imports]
testpkg="github.com/test/testpkg"

[[entity]]
    name="Person"
    [[entity.fields]]
        name="Name"
        type="string"
        description="Name"
    [[entity.fields]]
        name="PhoneNumber"
        type="testpkg.PhoneNumber"
        description="Phone Number"
    [[entity.fields]]
        name="Gender"
        type="bool"
        description="Male"
    [[entity.fields]]
        name="CreditCard"
        type="CreditCard"
        description="User who this card belongs to"
        relation="o2o"

[[entity]]
    name="CreditCard"
    [[entity.fields]]
        name="Number"
        type="string"
        description="Credit card number"
```

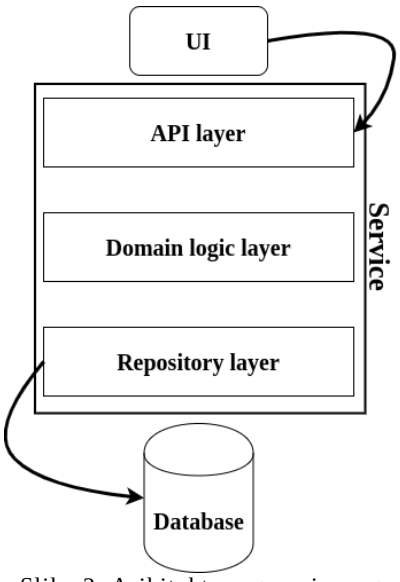
Listing 1. Izgled modela u TCG DSL-u

Generator servisa se koristi za generisanje serverske strane web aplikacije. Izlaz iz ove komponente je troslojna serverska aplikacija koja pruža HTTP API za obavljanje osnovnih CRUD operacija nad modelom podataka koji nastaje kao izlaz iz prethodne komponente. Generisanje se sastoji iz četiri faze, od kojih prva odgovara generisanju domenskih objekata, tj. modela podataka u generisanoj aplikaciji, a preostale tri faze odgovaraju slojevima aplikacije. Arhitektura izlazne aplikacije je prikazana na slici 2.

Generisanje klijentske aplikacije je posljednji korak u generisanju koda. Klijentska aplikacija se sastoji iz tri cjeline:

- prikaz zaglavlja za navigaciju
- prikaz liste entiteta
- forma za dodavanje, izmjenu ili pregled pojedinačnog entiteta.

Za svaki entitet generiše se posebna stranica kojoj se lako pristupa iz zaglavlja. Svaka stranica sastoji se iz liste entiteta i forme u kojoj se prikazuje, mijenja ili dodaje novi entitet. Stranice pojedinačnih entiteta su zadužene za komunikaciju sa API-jem za taj entitet koji pruža serverska aplikacija.



Slika 2. Arihitektura generisanog rješenja

3. IMPLEMENTACIJA RJEŠENJA

Generator je implementiran upotrebom programskog jezika Go. Kako su Go kompajler, kao i serverska aplikacija koja se generiše, napisani u istom tom jeziku, značajno je olakšano generisanje koda. Neke provjere, poput provjere tipova ili imenovanja polja i entiteta, implementirana su upotrebom koda koji se koristi i u samom kompajleru. Prilikom implementacije se izbjegavala upotreba dodatnih alata, kako bi se zadržala homogenost rješenja, olakšalo korištenje i smanjila zavisnost od spoljašnjih biblioteka ili alata.

3.1 Tehnologije

Kao što je već spomenuto, za implementaciju TCG generatora je korišten programski jezik Go, kao i tehnologije koje se na njega oslanjaju. Go šabloni su korišteni za generisanje koda. Izgenerisani serverski kod je pisan u Go programskom jeziku, a korišten je Gin [4] radni okvir. Baza podataka koju generisani servis koristi je PostgreSQL [5]. Za mapiranje objekata na relacioni model korišten je Gorm [6] ORM. Klijentska aplikacija koristi React JavaScript biblioteku. Za stilizovanje klijentske aplikacije je korištena Material-UI [7] biblioteka. Za pakovanje koda u kontejner koristi se Docker [8] alat.

3.2 TCG generator

Kako je generator napisan u Go programskom jeziku, neophodno je navesti nekoliko osobina standardnog seta alata ovog jezika, kao i neke njegove osobine na kojima se pojedine funkcionalnosti generatora i generisanog koda zasnivaju. Kao što je spomenuto, Go kompajler je pisan u Go jeziku, pa su neke od stadardnih biblioteka korištene za određene provjere prilikom generisanja modela. Takođe, standardni alati su korišteni za dobavljanje zavisnosti izgenerisanog koda, kao i za formatiranje izlaznih dokumenata. Još jedna važna osobina ovog jezika je to što dolazi sa ugrađenim alatom za obradu šablona (engl. *template engine*). To znači da je obrada šablona uprošćena i nazivasna od spoljašnjih alata i biblioteka, što značajno olakšava upotrebu samog generatora. Primjer Go šablona dat je na listingu 2.

```
{{ with .Entity }}
  {{ range . }}
    type {{ .Name }} struct {
      Model
       {{ with .Fields }}
         {{ range . }}
           {{ .Name }} {{ .Type }} `{{ .Tags }}`
             {{- end }}
          {{- end }}
      }

      type {{ .Name }}Page struct {
          Content []{{ .Name }} `json:"content"`
          Limit   int           `json:"limit"`
          Offset  int           `json:"offset"`
          Total   int           `json:"total"`
      }

      type {{ .Name }}Repository interface {
          Create({{ .Name }}) ({{ .Name }}, error)
          Read(string) ({{ .Name }}, error)
          List(int, int) ({{ .Name }}Page, error)
          Update({{ .Name }})({{ .Name }}, error)
          Delete({{ .Name }}) error
      }
  {{ end }}
{{- end }}
```

Listing 2. Primjer Go šablona

Za implementaciju sloja koji komunicira sa bazom podataka (*repository* sloj) koriti se GORM alat za objektno-relaciono mapiranje. Za te potrebe generišu se odgovarajući tagovi. Tagovi su meta-podaci koje ovaj alat koristi kako bi objekte sačuvao u bazi. Takođe, tagovi se koriste i od strane Gin radnog okvira za mapiranje JSON tijela HTTP zahtjeva na objekte i mapiranje objekata u JSON sadržaj HTTP odgovora.

Kao što je spomenuto, serverska aplikacija se sastoji iz tri sloja. Za svaki sloj se generiše Go interfejs sa odgovarajućim metodama. Go interfejs je imenovana kolekcija metoda. Za razliku od nekih drugih jezika, implementacija interfejsa je u Go programskom jeziku implicitna - svaki tip koji implementira sve metode iz date kolekcije implementira interfejs. Na taj način je generisana implementacija razdvojena od specifikacije slojeva serverske aplikacije, tj. kao zavisnosti između slojeva se prenose interfejsi, a ne njihova implementacija. Ovaj pristup pruža veliku fleksibilnost i olakšava izmjene generisane implemntacije, upotrebu korisničke implementacije, a omogućuje i kombinaciju generisanog i korisničkog koda. Na listingu 2 je prikazan generisani kod koji koristi tagove i interfejse.

Za integraciju sa ručno pisanim kodom ili proširenje generisane implementacije se može koristiti tzv.

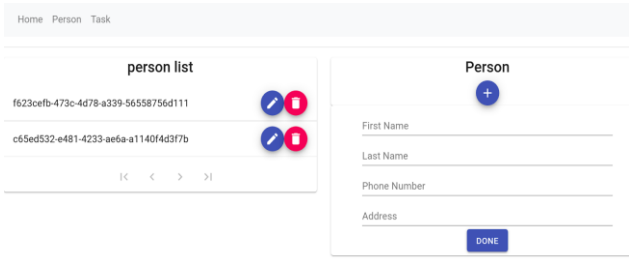
middleware mehanizam. Ovaj mehanizam podrazumijeva da korisnički kod implementira generisane interfejse, ali da kao zavisnost koristi generisanu implementaciju. Tako korisnički kod može da izvršava željene radnje prije ili poslije poziva generisanog koda, a moguće je, ukoliko korisnik to želi, potpuno izbjeći upotrebu generisanog koda i zamijeniti je korisničkom.

```
type Person struct {
    Model
    Name      string `json:"name" form:"name"`
    Phone     string `json:"phone" form:"phone"`
    Address   string `json:"address" form:"address"`
    Task      Do `json:"task" gorm:"foreignkey:pid"`
}

type PersonRepository interface {
    // Create creates a new Person.
    Create(Person) (Person, error)
    // Read reads an existing Person.
    Read(string) (Person, error)
    // List returns a page of Person.
    List(int, int) (PersonPage, error)
    // Update updates an existing Person.
    Update(Person) (Person, error)
    // Delete removes an existing Person.
    Delete(Person) error
}
```

Listing 3. Upotreba tagova i interfejsa u generisanom kodu

Klijentska aplikacija je napisana u JavaScript programskom jeziku pomoću React biblioteke. Kod koji TCG generiše koristi JSX (jezik koji je kombinacija JavaScript i HTML) za prikaz elemenata i Material-UI za stilizovanje, pa je moguće generisani kod za jedan entitet smjestiti u jednu komponentu. React biblioteka podržava komponentni dizajn. Svaka kompenenta ima stanje (engl. *stateful component*) koje se mijenja u skladu sa korisničkom aktivnošću. Svaki entitet je predstavljen jednom komponentom koja komunicira sa krajnjim tačkama na serverskoj strani i poziva CRUD operacije za dati entitet. Pored ovih, postoji još jedna komponenta koja služi za navigaciju na korisničkom interfejsu. Svaki ekran za prikaz entiteta se sastoji iz dva dijela - dio za prikaz liste entiteta i forma za prikaz, izmjenu ili dodavanje pojedinačnog entiteta. Na slici 3 je prikazan izgled generisane klijentske aplikacije.



Slika 3. Izgled klijentske aplikacije

4. ZAKLJUČAK

U radu je opisan TCG generator serverske i klijentske web aplikacije. Za potrebe tog generatora kreiran je mini-jezik kako bi se što više uprostilo pisanje ulaznog dokumenta i korištenje generatora. Kod je generisan sa namjerom da bude lako čitljiv i razumljiv. Podržana je integracija sa ručno pisanim kodom, kao i izmjene i prilagođavanje izgenerisanog koda. Različiti slojevi aplikacija su razdvojeni i nezavisni, pa su i prilagođavanje i izmjene istih međusobno izolovani, što smanjuje mogućnost greške. Primarna namjena TCG generatora je upotreba u brzom prototipskom razvoju softvera. Jednostavnost upotrebe i korištenje kontejnerizacije generisanog koda pružaju mogućnost brzog razvoja, značajno skraćujući put od ideje do realizacije. Generisana klijentska aplikacija, iako generička, pruža lak uvid u generisane entitete kojima se modeluje neki stvarni sistem, kao i API koji serverska aplikacija pruža. U daljem razvoju je planirano dodavanje validacije HTTP zahtjeva, odnosno validacije modela. Takođe, dodatno unapređenje bi bilo i dodavanje autorizacije i kontrole pristupa, što je standardna funkcionalnost modernih web aplikacija.

5. LITERATURA

[1] TOML: Tom's Obvious, Minimal Language, https://github.com/toml-lang/toml, pristupljeno 12.06.2020.

[2] Roy Thomas Fielding, Architectural Styles and the Design of Network-based Software Architectures, University of California, Irvine, CA, USA, 2000.

[3] YAML: YAML Ain't Markup Language, https://yaml.org/, pristupljeno 12.06.2020.

[4] Gin Web Framework, https://github.com/gin-gonic/gin, pristupljeno 13.06.2020.

[5] PostgreSQL, https://www.postgresql.org/, pristupljeno 13.06.2020.

[6] GORM, https://gorm.io/, pristupljeno 13.06.2020.

[7] Material-UI, https://material-ui.com/, pristupljeno 13.06.2020.

[8] Docker, https://www.docker.com/, pristupljeno 13.06.2020.

Biografija:

Dušan Borovčanin rođen je 31.03.1994. godine u Sokocu. Osnovnu školu „Sokolac“, u Sokocu, završio je 2009. godine. Gimnaziju „Vasilije Ostroški“ u Sokocu završio je 2013. godine. Iste godine upisao je Fakultet tehničkih nauka u Novom Sadu, smjer Softversko inženjerstvo i informacione tehnologije. Osnovne akademske studije završio je u oktobru 2017. godine. Iste godine je upisao master akademske studije na smjeru Elektronsko poslovanje.