Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.42:004.738.12
DOI: https://doi.org/10.24867/23BE10Petakovic

JEZIK ZA OPIS JEDINIČNIH TESTOVA BESERVERSKIH APLIKACIJA

DOMAIN-SPECIFIC LANGUAGE FOR DESCRIBING SERVERLESS UNIT TESTS

Aleksandar Petaković, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *Ovaj rad predstavlja i opisuje jezik za specifikaciju testova za beserverske aplikacije. Opisana je i implementacija sistema koji interpretira dati jezik i generiše testove na drugim programskim jezicima na osnovu šablona. U sklopu sistema, razvijena je i nadogradnja za Visual Studio Code za naglašavanje sintakse prilikom opisa testova.*

Ključne reči: *DSL, beserverske tehnologije, jedinični testovi, textX, Visual Studio Code*

Abstract – *This paper presents and introduces a language for specifying unit tests for serverless applications. Also described is the implementation process of the system that interprets this language and generates tests for other programming languages based on templates. As a part of the system, a Visual Studio Code extension for syntax highlighting was also developed.*

Keywords: *DSL, serverless, unit tests, textX, Visual Studio Code*

1. UVOD

Jezici specifični za domen (skraćeno JSD, eng. *Domain-Specific Languages*) su počeli da se razvijaju kad i sami programski jezici, a razvoj jezika specifičnih za domen najčešće obuhvata proučavanje domena, konsultacije sa stručnjacima i velike količine planiranja i projektovanja. Sa druge strane spektra, u prethodnih deset godina razvijale su se tehnologije koje su danas nezaobilazne kada se priča o razvoju internet aplikacija i koje omogućavaju brz i lak pristup tim aplikacijama širom sveta, od strane miliona korisnika. Te tehnologije se mogu jednim imenom nazvati beserverskim (eng. *serverless*) tehnologijama.

Testiranje beserverskih aplikacija, kao i testiranje svih drugih tipova softvera, najčešće obuhvata pisanje programskog koda koji će iskoristiti delove softvera i proveriti da li su rezultati upotrebe tih delova zadovoljavajući. Obim testova često ume da bude približan, a nekada i da prevazilazi obim softvera koji se testira, pa se može zaključiti da je pisanje testova, odnosno samo testiranje softvera jedan zahtevan posao. U ovom radu biće predstavljeno rešenje čija je osnovna ideja ubrzanje i olakšanje pisanja testova za beserverske aplikacije.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Igor Dejanović, red. prof.

Sistem će se bazirati na pogodnostima i prednostima razvoja jezika specifičnog za domen, za rešavanje problema ovog uskog domena.

2. TEORIJSKA OSNOVA

Tri glavne oblasti na kojima se bazira čitav rad i koje će biti predstavljene u najviše detalja u nastavku jesu: jezici specifični za domen, beserverske aplikacije, i testiranje softvera.

2.1. Jezici specifični za domen

Programski jezik je veštački jezik čija je svrha upravljanje računarom. Programski jezici su, slično ljudskim jezicima, definisani kroz sintaksna i semantička pravila, kako bi se odredila njihova struktura i njihovo značenje [1]. Ako posmatramo tip problema kao oblast znanja, odnosno sferu interesa, primetićemo da se pojedini programski jezici mogu protumačiti i kao jezici prilagođeni tipu problema, odnosno jezici specifični za domen (problema).

Radi utemeljivanja osnovih koncepata, baziraćemo se na definiciji jezika specifičnih za domen koju je postavio Martin Fauler (eng. *Martin Fowler*) u svojoj knjizi o jezicima specifičnim za domen: „*Jezici specifični za domen su programski jezici ograničene ekspresivnosti, fokusirani na određeni domen*“ [2].

2.2. Beserverske aplikacije

Beserverske tehnologije predstavljaju vid razvoja softvera u oblaku koji omogućava razvojnim timovima da razvijaju i pokreću svoja softverska rešenja bez upravljanja serverima. Kao početak ere beserverskog razvoja softvera, najčešće se uzima početak upotrebe servisa Lambda razvijenog od strane američke kompanije Amazon, koji je pušten u javnost krajem 2014. godine [3]. Razvojni timovi su ubrzo osetili prednosti ovog pristupa koji im je omogućio da se fokusiraju na kod, odnosno srž svakog softvera, bez da se brinu i troše vreme na kupovinu i podešavanje hardvera, operativnog sistema i ostalog pratećeg softvera na kome će se kod izvršavati.

2.3. Testiranje softvera

Pri razvoju softvera, potrebno je utvrditi da li softver zadovoljava određene zahteve, a stepen usklađenosti sa zahtevima direktno određuje kvalitet softvera. Kao definiciju testiranja softvera, možemo iskoristiti sumu glavnih teza o testiranju softvera: Testiranje je proces koji se sastoji od svih aktivnosti u životnom ciklusu softvera vezanih za planiranje, pripremu i izvršavanje zadataka koji treba da pokažu da li softver zadovoljava specificirane zahteve i ispunjava namenu, kao i da ukažu na greške u softveru [4].

3. SPECIFIKACIJA SISTEMA

Sistem, i jezik na kome se bazira čitav sistem, nose naziv „ServerlessTestS“, ili skraćeno STS.

3.1 Motivacija i cilj rešenja

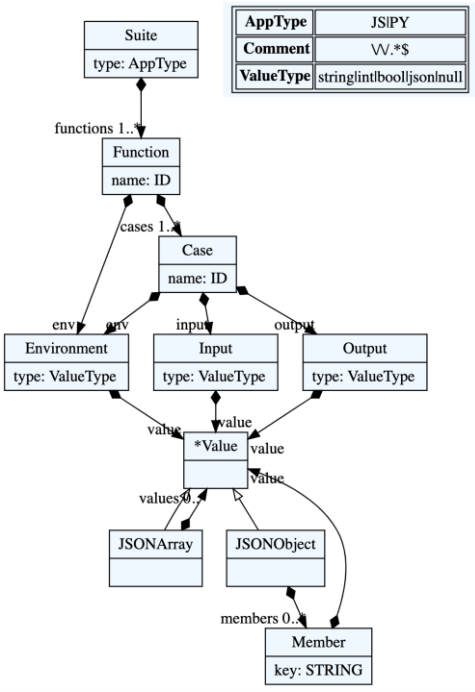
U svetu razvoja beserverskih aplikacija, kao i u ostalim sferama razvoja softvera, potrebno je testirati svako rešenje koje je namenjeno za upotrebu od strane drugih korisnika. Postoji više pristupa u testiranju beserverskih aplikacija, međutim, kada je u pitanju sama implementacija, ovi pristupi nisu zahvalni iz više razloga. Cilj ovog jezika, odnosno sistema, jeste da prevaziđe ove nedostatke i ponudi rešenje koje bi bilo lakše za savladati od ostalih programskih jezika. Zatim, koje bi nudilo dovoljno opcija za većinu slučajeva koji se testiraju, i koje ne bi zahtevalo pisanje velike količine šablonskog koda. Ovakav pristup je moguć uz korišćenje jezika specifičnog za domen kao alata za modelovanje testova, a zatim generisanje koda testova na određenom programskom jeziku.

3.2 Model sistema

Rešenje koje ovaj rad predstavlja, *ServerlessTestS*, sastoji se, pre svega iz istoimenog jezika specifičnog za domen. *ServerlessTestS* jezik razvijen je kao eksterni jezik specifičan za domen, sa ciljem da bude pregledan, ali i funkcionalan - i omogućava modelovanje testova za koje želimo da generišemo programski kod na nekom od podržanih jezika. Modelovanje se vrši upotrebom elemenata jezika uz specificiranje proizvoljnih podataka gde je to neophodno. Zatim, na osnovu gramatike ovog jezika, parser STS jezika parsira specifikaciju testova koju mu prosleđujemo. Parser vrši inicijalnu validaciju modela, a kao rezultat parsiranja dobijamo strukturu objekata. Ova struktura objekata je povoljan oblik podataka za dalju obradu i generisanje testova na osnovu šablona, a pre svega i validaciju.

3.3 ServerlessTestS jezik

Prikaz elemenata *ServerlessTestS* jezika i veze između istih, prikazan je na slici 1.



Slika 1: *Model ServerlessTestS jezika*

Kako su podržani i *JavaScript* i *Python* jezici prilikom generisanja programskog koda testova, teško je bilo napraviti paralelu između sintakse ova dva jezika. Međutim, beserverske aplikacije su najčešće veb-bazirane aplikacije, pa možemo pretpostaviti da se većina razvojnih timova koji se bave izradom veb- baziranih aplikacija susretala sa *markup* jezicima, naime XML-om, ili HTML-om. Stoga je STS sintaksa i gramatika razvijana tako da podseća na *markup* jezike.

4. IMPLEMENTACIJA

Svrha ovog poglavlja jeste dublje upoznavanje sa ServerlessTestS jezikom i sistemom, kao i predstavljanje procesa razvoja sistema.

4.1 Tehnologije

Alat upotrebljen za izradu i implementaciju STS jezika jeste *textX*, dok su za uspešno izvršavanje generisanih testova upotrebljeni elementi radnh okvira za testiranje, odnosno – *Jest* kada je u pitanju *JavaScript*, i *pytest*, kada je u pitanju *Python*. A za generisanje programskog koda upotrebljen je *jinja* obrađivač šablona.

TextX je meta-jezik, odnosno jezik za definisanje jezika, čija svrha je specifikacija jezika specifičnih za domen kroz *Python* programski jezik.

Jinja je obrađivač šablona (eng. *Template engine*) implementiran u *Python* programskom jeziku.

4.2 ServerlessTestS jezik

Suite je osnovni element *ServerlessTestS* jezika koji sadrži sve druge elemente. Jezik je zamišljen tako da jedna „sts“ datoteka predstavlja jednu instancu *suite* elementa, što bi u svetu testiranja softvera grubo odgovaralo nizu povezanih, odnosno sličnih testova.

Function element je jedan od glavnih elemenata *ServerlessTestS* jezika. Ideja ovog elementa jeste da sadrži sve testove vezane za jednu beserversku funkciju. Takođe može sadržati i element za definisanje promenljivih okruženja.

Environment element je zamišljen tako da se kroz njega mogu definisati sve neophodne promenljive okruženja koje su potrebne za izvršavanje testova.

Case element služi za definisanje test slučajeva vezanih za jednu funkciju. Slično *Function* elementu – i *Case* elementu je potrebno dodeliti ime kao identifikator elementa i to ime mora biti jedinstveno na nivou jednog *Function* elementa.

Input i *Output* elementi služe za navođenje ulaznih, odnosno izlaznih podataka prilikom modelovanja testova.

4.3 Interpretacija parsiranih modela

U ovom sistemu, pre generisanja koda, vrši se neophodna provera parsiranih modela, gde se proveravaju pretežno semantička ograničenja.

Za primenu ovih ograničenja, iskorišćena je funkcionalnost procesora koju nudi *textX* alat. Upotrebljene su dve vrste procesora, procesori modela, i procesori objekata.

4.4 Generisanje koda

Šablon za generisanje *JavaScript* testova je implementiran tako da koristi funkcionalnosti *Jest* radnog okvira za testiranje, dok šablon za generisanje *Python* testova koristi funkcionalnosti *pytest* radnog okvira za testiranje.

Glavni izazov u oba šablona bio je pravilno formatiranje JSON objekata. Za rešavanje ovog problema, upotrebljena je funkcionalnost *jinja*-e za formiranje makroa.

4.5 Komanda za generisanje testova

U cilju lakog pokretanja sistema, formirana je komanda pri čijem izvršavanju će biti izvršeno generisanje testova, a kojoj je moguće proslediti putanje do željenog ulaza i izlaza.

4.6 Nadogradnja za sintaksno naglašavanje

Kako bi korisnici imali što bolje iskustvo pri korišćenju STS jezika i formiranju modela testova. Kako je Visual Studio Code (skraćeno VSCode), razvojno okruženje i editor teksta razvijen od strane *Microsoft*-a - jedno od trenutno najpopularnijih razvojnih okruženja, u kome je i sam STS jezik razvijan – implementirana je nadogradnja upravo za ovaj editor.

5. KONFIGURISANJE I IZVRŠAVANJE SISTEMA

Ovo poglavlje biće posvećeno konfigurisanju i izvršavanju samog sistema i prikazu rezultata izvršavanja sistema kroz primer modelovanja testova za dve beserverske aplikacije.

5.1 Konfigurisanje sistema

Čitav sistem je napisan u *Python* programskom jeziku. A kako bi sistem bio prenosiv, i instalacija i konfiguracija tekla lakše na ostalim računarima, sistem je konfigurisan tako, da se može preneti i instalirati na željenom računaru uz *pip*, upravljač *Python* paketima. Nakon uvezivanja, odnosno instalacije uz pomoć *pip* upravljača paketima, funkcionalnosti sistema će biti dostupne kroz *textX* CLI (eng. *Command Line Interface* – interfejs komandne linije).

5.2 Konfigurisanje nadogradnje za VSCode

VSCode nudi šablon za kreiranje nadogradnji i taj šablon je iskorišćen za kreiranje ove nadogradnje. Nadogradnje za VSCode kreirane na ovaj način, koje takođe nisu objavljene na zvanični repozitorijum nadogradnji, mogu lokalno da se uvežu premeštanjem direktorijuma nadogradnje u direktorijum rezervisan za VSCode nadogradnje.

5.3 Generisanje i izvršavanje testova

U cilju da se što bolje predstavi prenosivost STS koda između dva jezika, *JavaScript* i *Python* beserverska aplikacija implementiraju iste funkcionalnosti. Funkcije specificirane u aplikacijama za koju će biti generisani testovi, prikazane su na slici 2.

Specificirana su dva test slučaja za testiranje funkcije *hello*. Prvi slučaj testira pozitivan ishod funkcije, gde je promenljiva okruženja definisana i gde su *header*-i zahteva ispravni. A u drugom slučaju očekujemo negativan ishod izvršavanja zbog prisustva simbolično nazvanog „lošeg“ (*Bad* u prevodu na engleski jezik) *header*-a.

Kada je u pitanju funkcija *activity*, u mogućnosti smo samo da testom potvrdimo da će izvršavanje funkcije kao rezultat vratiti HTTP odgovor sa status kodom 200.

```
functions:
  hello:
    handler: hello.handler
    events:
      - http:
          path: hello
          method: get

  activity:
    handler: activity.handler
    events:
      - http:
          path: activity
          method: get

  calculate:
    handler: calculate.handler
    events:
      - http:
          path: activity
          method: get

```

Slika 2: *Funkcije specificirane u serverless.yml*

Za funkciju *calculate* ponovo imamo specificirana dva slučaja. U prvom slučaju vršimo operaciju množenja nad brojevima dva i tri, a u drugom slučaju sabiramo ta dva broja. Kao rezultat izvršavanja prvog slučaja, očekujemo broj šest, dok kao rezultat izvršavanja drugog slučaja, očekujemo broj pet. Željene operacije prosleđujemo kroz promenljivu okruženja „OPERATION“. Specifikacija prvog slučaja prikazana je na slici 3.

```
< function = calculate
  < case = calculateCase1
    < env = json
      {
        "OPERATION": "multiply"
      }
    >
    < input = json
      {
        "a": 2,
        "b": 3
      }
    >
    < output = int
      6
    >
  >
```

Slika 3: *Funkcija calculate, prvi test slučaj*

Generisanje testova moguće je izvršiti pokretanjem komande „textx sts-generate” kroz interfejs komandne linije *textX*-a. Rezultat generisanja su tri datoteke koje nose nazive funkcija, uz nastavke specifične za okruženje, odnosno jezik za koje su generisani.

5.4 Pregled rezultata izvršavanja sistema

Otvaranjem bilo koje generisane datoteke, primetićemo običan *JavaScript*, odnosno *Python* kod, međutim, ono što možda nije očigledno na prvi pogled, jeste odnos broja karaktera specifikacije i samih testova. Ako saberemo broj karaktera u generisanim datotekama, i uporedimo ga sa brojem karaktera u STS specifikaciji, primetićemo brojeve navedene u tabeli 1.

Tabela 1: *Broj karaktera potreban za specifikaciju testova*

Jezik	JavaScript	Python	ServerlessTestS
Broj karaktera	1823	2054	1440

Broj karaktera potreban za specifikaciju testova na *JavaScript* jeziku je 27% veći, dok je broj karaktera potreban za specifikaciju testova na *Python* jeziku čak 43% veći.

5.5 Primena

Primarno se sistem može upotrebiti za specificiranje i generisanje testova tokom razvoja beserverske aplikacije.

Upotrebom ovog sistema, bilo bi moguće skladištiti samo specifikaciju testova na repozitorijumu, a generisanje testova bi moglo da se odvije tek kada je to neophodno da bi se izvršili testovi, odnosno, u CI (eng. *Continuous Integration*) okruženju.

Direktna posledica čuvanja samo specifikacije testova umesto svih datoteka koje se mogu generisati jeste i bolja, odnosno veća prenosivost koda između dva uređaja, ili uređaja i repozitorijuma.

5.6 Smernice za dalji razvoj sistema

Glavne smernice za unapređenje ovog sistema, bile bi unapređivanje nadogradnje za VSCode i dodavanje podrške za više editora, zatim podrška za sekcije ručno unesenog koda. Sledeće, podrška za više kriterijuma za prolaznost testova, takođe i integracija sa beserverskim radnim okvirom i podrška za više jezika.

6. ZAKLJUČAK

Priloženo rešenje opisano kroz ovaj rad nudi mogućnost efikasnijeg specificiranja testova za beserverske aplikacije. Ovo rešenje takođe demonstrira moć koju imaju jezici specifični za domen, na kojima se zasniva, i pokazuje da i dalje u nekim domenima postoji prilika za kreiranje jezika koji bi približio domen korisnicima. Takođe, u situacijama u kojima se pojavljuju linije koda koje se ponavljaju, treba uvideti i kreirati šablon, pre svega kako bi se izbegle greške prilikom kopiranja i lepljenja delova koda, ali i kako bi se moglo automatizovati popunjavanje šablona, kao što je slučaj u ovom sistemu. Ukupan rezultat ovih rešenja je efikasniji proces izrade novih softverskih rešenja i manje prostora za greške. A to je ujedno i budućnost izrade softvera kakvoj treba težiti.

7. LITERATURA

[1] Programming language - https://www.cs.mcgill.ca/~rwest/wikispeedia/wpcd/wp/p/Programming_language.htm (pristupljeno u julu 2022.)

[2] Martin Fowler, Rebecca Parsons, „Domain Specific Languages“, Addison-Wesley Professional, 2010.

[3] AWS Lambda Wikipedia - https://en.wikipedia.org/wiki/AWS_Lambda (pristupljeno u novembru 2022.)

[4] Andreas Spillner, Hans Schaefer, Tilo Linz, „Software Testing Foundations: A Study Guide for the Certified Tester Exam“, Rocky Nook, 2021.

Kratka biografija:

Aleksandar Petaković rođen je u Sremskoj Mitrovici 1997 god. Po završetku srednje škole, 2016. godine, upisuje smer računarstvo i automatika na Fakultetu Tehničkih Nauka u Novom Sadu. Diplomirao je 2020. godine, nakon čega je iste godine upisao master studije.

kontakt:
aleksandar.petakovic@outlook.com