Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/21BE08Savic

AUTOMATIZOVANO TESTIRANJE KORISNOSTI VEB APLIKACIJA POMOĆU JEST OKVIRA

AUTOMATED USABILITY TESTING OF WEB APPLICATIONS USING THE JEST FRAMEWORK

Aleksandar Savić, *Fakultet tehničkih nauka, Novi Sad*

Oblast – RAČUNARSTVO I AUTOMATIKA

Kratak sadržaj – *U ovom radu je opisan Jest alat za automatizovano RESTful API testiranje veb aplikacije. Na studiji slučaja aplikacije za manipulaciju korisnika je ilustrovano korišćenje ovog alata za automatizovano testiranje realnih veb aplikacija. Objašnjeni i implementirani su API testovi za svaku funkcionalnost aplikacije.*

Ključne reči: *Jest, automatizovano testiranje, API, REST, Express*

Abstract – *This paper describes the Jest tool for automated RESTful API testing of a web application. A user manipulation application case study illustrates the use of this tool for automated testing of real web applications. API tests for each application functionality are explained and implemented.*

Keywords: *Jest, automated testing, API, REST, Express*

1. UVOD

Testiranje softvera je važan deo razvoja svih aplikacija. Za veb aplikacije, međutim, to je neizbežno. To je zbog činjenice da je razvoj veb aplikacija skup i dugotrajan, ali pažljivo biranje poboljšanih metoda testiranja dovodi do jeftinijeg razvoja i održavanja [1]. Jedan od načina za smanjenje troškova razvoja i testiranja aplikacija je automatizovano testiranje. Sadašnji razvoj skripti za automatizovano testiranje veb aplikacija, međutim, pati od mnogih nedostataka. Stoga, često mala promena u aplikaciji dovodi do mnogih promena u testovima, a samim tim pruža više mogućnosti za unošenje grešaka u test. Razvoj i održavanje takvih testova je stoga skupo [2]. Zato je Jest predložen kao idealan alat za automatizaciju testova.

2. AUTOMATIZOVANO TESTIRANJE

Automatizovano testiranje predstavlja metodu kojom se test procedure automatizuju pomoću nekog alata za izvršavanje testova. Na ovaj način, test koji se jednom isplanira i za koji se kreira skripta može se uvek pustiti bez trošenja vremena na ponavljanje koraka testa. Za automatizovano testiranje postoji preduslov da tester poseduje znanje nekog programskog jezika.

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

2.1. Motivacija

Tradicionalno, proces testiranja softvera je uglavnom bio ručni. Promene koda koje je napravio jedan programer je pregledao drugi, a sam softverski proizvod je prošao ručno regresijsko testiranje od strane timova za osiguranje kvaliteta (QA), tek nakon što je proces razvoja završen. Ručni testeri su pratili softversku dokumentaciju korak po korak kako bi potvrdili da nove i postojeće funkcionalnosti rade kako je definisano. Svi problemi koje je otkrio QA su prosleđeni nazad programerima da ih poprave. Kada je testiranje softvera odvojeno od razvoja i sprovodi se samo nekoliko puta mesečno ili godišnje, programeri često uče o greškama koje su napravili nedeljama ili mesecima nakon što su ih napravili.

Do tog trenutka teško je setiti se logike koda koji ima grešku, a greška postaje spora i teško ju je rešiti i iz nje učiti. Osim sporih povratnih informacija za programere, ručno testiranje usporava tempo kojim nove verzije softvera mogu biti objavljene klijentima. Pouzdanost ručnog pristupa takođe se može dovesti u pitanje, pošto su ručni zadaci koji se ponavljaju skloni ljudskim greškama. Štaviše, kako softverski sistem koji se testira (SUT) raste i evoluira, dokumentacija koja se koristi za ručno testiranje takođe zahteva da se stalno ažurira, što košta dodatno vreme i trud.

Automatizovano testiranje rešava gore opisane probleme. Danas je široko prihvaćena praksa da programeri pišu automatizovane testove koji ne uspeju u izgradnji aplikacije ako otkriju regresije. Lokalno pokretanje automatizovanih testova omogućava programerima da rano primete greške i da ih brzo poprave, često pre nego što se greške provere u kontroli verzija.

2.2. Tipovi automatizovanih testova

Automatizovano testiranje softvera može se vršiti na različitim nivoima, od najjednostavnijih gradivnih blokova aplikacije do načina na koji aplikacija funkcioniše kao celina. Test piramida, koncept koji je popularisao Kon, je način vizuelizacije strategije testiranja koja uključuje više slojeva različitih tipova automatizovanih testova.

Piramida obezbeđuje ravnotežu brzine i složenosti postavljajući veliki broj malih testova koji se brzo izvršavaju na dnu, sa sve sporijim i složenijim testovima ka vrhu. Tri tipa automatizovanih testova pomenutih u piramidi su testovi jedinica, integracije i end-to-end testovi.

Postoji više terminologija koje definišu različite vrste automatizovanih testova.

2.3. Ciljevi test automatizacije

Testovi bi trebalo da poboljšaju kvalitet softvera. Kvalitet softvera se može utvrditi odgovorom na dva pitanja: da li je softver pravilno napravljen i da li je napravljen pravi softver. Praksa koja pomaže da se odgovori na ova pitanja nezavisno jedno od drugog je pisanje testova pre pisanja softvera. Ovaj pristup olakšava razmatranje šta softver treba da radi odvojeno od načina na koji to treba da radi. Pristup pisanju testova kao prvi test, gde testovi pomažu u dizajniranju softvera, široko je rasprostranjen u industriji. Prvo pisanjem testova lako je verifikovati ponašanje sistema pre verifikacije njegove implementacije.

Test-driven development je tehnika razvoja softvera gde se proces razvoja vodi pisanjem testova. Svaki put kada treba implementirati novi deo funkcionalnosti, prvo se napiše test za tu funkcionalnost. Zatim se sama funkcija implementira i prolazi test. Konačno, i karakteristika i test se refaktorišu radi strukture i jasnoće. Pisanje testova pre softvera daje jasnu definiciju uspeha. Na ovaj način, testovi deluju kao specifikacija za softver.

2.4. Pokrivenost koda

Pokrivenost koda je metrika koja se koristi za izražavanje koje delove softvera testovi ne koriste, koliki deo izvornog koda je testiran. Ova metrika se može primeniti na različite vrste automatizovanih testova uključujući testiranje jedinica.

Iako 100% pokrivenost testom može izgledati kao dobar cilj, u stvarnosti to ne garantuje nedostatak nedostataka. Pokrivenost od 100% samo znači da je sav kod koji se testira bio izvršen, a ne da se sistem ponaša u skladu sa specifikacijom.

2.5. Razvoj vođen ponašanjem i testiranje u stilu specifikacije

Razvoj vođen testovima se zalaže za korišćenje jedne pune rečenice za svaki naziv metode testiranja. Korišćenje punih rečenica omogućava izradu softverske dokumentacije koja ima smisla ne samo za programere, već i za menadžere projekata, analitičare, testere i druge poslovne korisnike. Na ovaj način BDD može da obezbedi ažurnu dokumentaciju za softver.

Automatizovani testovi se proveravaju u kontrolu verzija zajedno sa kodom aplikacije, a programeri koji žele da razumeju kako sistem funkcioniše mogu da pročitaju test paket kao primer kako da koriste API sistema. Držanje jedne rečenice po nazivu testa čini testove fokusiranijim, jer se samo relativno mali deo ponašanja može opisati u rečenici. Nazivi rečenica su takođe od pomoći u slučajevima kada testovi ne uspeju – željeno ponašanje je odmah jasno.

2.6. Test doubles

Softver koji se testira sastoji se od mnogo jedinica. Svaka jedinica ima određeno ponašanje. Neke jedinice su sadržane u jednoj klasi ili funkciji. Međutim, većina jedinica zahteva interakciju sa drugim jedinicama. Prilikom testiranja određene jedinice, ovo oslanjanje na druge jedinice (koje se nazivaju saradnici) može biti problematično – ili saradnik nije dostupan, ili saradnik vraća rezultate koji nisu prikladni za test, ili izvršavanje saradnika ima nepotrebne nuspojave. U ovim situacijama, saradnici mogu biti zamenjeni kako bi se stekla potpuna kontrola nad okruženjem u kojem se testira jedinica. Test double je alat koji se koristi za izolaciju jedinice od njenih saradnika kako bi se testirala.

Postavljanje testnih duplikata je obično prvi korak u testu. Uspostavljeni obrazac za strukturiranje testova jedinica je raspored-postupi-potvrđivati. Prvo se postavljaju objekti potrebni za test. Zatim se izvršava ponašanje koje se testira. Na kraju, daju se tvrdnje o rezultatu izvršenja. Isti obrazac je takođe opisan kao četvorofazni test. Četiri koraka su podešavanje, vežbanje, verifikacija i demontaža.

3. REST

Termin REST je skraćenica za Representational State Transfer i definisao ga je Roi Thomas Fielding u svojoj doktorskoj disertaciji: „Arhitektonski stilovi i dizajn mrežnih softverskih arhitektura“. objavljen 2000. REST nije sama softverska arhitektura, već „koordinirani skup arhitektonskih ograničenja koja pokušava da minimizira kašnjenje i mrežnu komunikaciju, dok maksimizira nezavisnost i skalabilnost implementacije komponenti”. Ovo poglavlje predstavlja REST: uvod u to kako Fielding pristupa REST-u kao arhitektonskom stilu, opis specifičnih ograničenja i elemenata koji čine REST, apstrakcije RESTful sistema i konačno, objašnjenje RESTful sistema koristeći HTTP.

3.1. REST pristup

REST je arhitektonski stil za distribuirane hipermedijske sisteme. To je hibridni stil izveden iz različitih arhitektonskih stilova zasnovanih na mreži i kombinovan sa dodatnim ograničenjima kako bi se definisao uniformni interfejs konektora. Fokusira se na ograničenja koja se moraju postaviti na semantiku konektora, dok se drugi stilovi fokusiraju na ograničenja semantike komponenti.

Dodatna ograničenja REST-a potiču od sledećih arhitektonskih stilova: ClientServer, Client-Stateless-Server, Client-Cache-StatelessServer, Laiered Sistem i Code On Demand. REST takođe uključuje koncepte resursa i uniformnog interfejsa.

3.2. Ograničenja

REST je arhitektonski stil za distribuirane hipermedijske sisteme. To je hibridni stil izveden iz različitih arhitektonskih stilova zasnovanih na mreži i kombinovan sa dodatnim ograničenjima kako bi se definisao uniformni interfejs konektora. Fokusira se na ograničenja koja se moraju postaviti na semantiku konektora, dok se drugi stilovi fokusiraju na ograničenja semantike komponenti.

Dodatna ograničenja REST-a potiču od sledećih arhitektonskih stilova: *ClientServer, Client-Stateless-Server, Client-Cache-StatelessServer, Laiered Sistem i Code On Demand.* REST takođe uključuje koncepte resursa i uniformnog interfejsa.

Interfejs Klijent-Server zahteva postojanje klijentske komponente koja šalje zahteve i serverske komponente koja prima zahteve i može da izda odgovor. Ovo ograničenje je zasnovano na principu razdvajanja briga. Jedinstveni interfejs razdvaja interfejse klijenata i servera. Ovo razdvajanje briga znači da klijenti, na primer, nisu povezani sa skladištenjem podataka koje je problem servera, a serveri nisu povezani sa korisničkim interfejsom ili korisničkim stanjem koje se tiče klijenta. Ovo poboljšava prenosivost interfejsa na više platformi i skalabilnost pojednostavljujući komponente servera. Takođe podržava nezavisnu evoluciju logike na strani klijenta i logike na strani servera, jer se

svaka komponenta može zameniti i razvijati zasebno sve dok se interfejs među njima ne menja. Klijent-Server je možda najosnovnije ograničenje jer se sva druga ograničenja pozivaju na njegove artefakte i tako se nadograđuju na ovo ograničenje.

Statelessness ograničenje se dodaje ograničenju Klijent-Server. U svakoj interakciji, komunikacija između klijenta i servera mora biti bez stanja. To znači da svaki zahtev bilo kog klijenta treba da sadrži sve informacije neophodne kako bi server razumeo značenje zahteva. Zatim, sve podatke koji se tiču stanja sesije treba vratiti klijentu. Dakle, stanje sesije se u potpunosti čuva na klijentu i server ne može ponovo da koristi informacije iz prethodnih zahteva.

Da bi se ublažilo smanjenje performansi mreže zbog ograničenja bez stanja, dodato je još jedno ograničenje na stil klijent-bez stanja-server prikazan ranije. REST uključuje ograničenje keša tako da naredni zahtevi serveru ne moraju da se upućuju ako su potrebni podaci već u lokalnom kešu na strani klijenta. Tako se formira klijent-keš-server bez stanja.

3.3. REST elementi

REST arhitektonski stil ograničava arhitekturu na arhitekturu klijent/server i dizajniran je da koristi komunikacioni protokol bez stanja, obično HTTP. U stilu REST arhitekture, klijenti i serveri razmenjuju reprezentacije resursa korišćenjem standardizovanog interfejsa i protokola. Svaka softverska arhitektura je sastavljena od komponenti, konektora i podataka. „REST ignoriše detalje implementacije komponenti i sintakse protokola kako bi se fokusirao na uloge komponenti, ograničenja njihove interakcije sa drugim komponentama i njihovu interpretaciju značajnih elemenata podataka“.

Uloga REST komponenti je uspostavljanje komunikacije. Korisnički agent koristi klijentski konektor da pokrene zahtev i postaje krajnji primalac odgovora. Izvorni server koristi serverski konektor da upravlja prostorom imena za traženi resurs. Proksi je posrednik koji je izabrao klijent da obezbedi inkapsulaciju interfejsa drugih usluga, prevod podataka, poboljšanje performansi ili bezbednosnu zaštitu. Mrežni prolaz je posrednik koji nameće mreža ili izvorni server da bi obezbedio inkapsulaciju interfejsa drugih usluga, za prevođenje podataka, poboljšanje performansi ili sprovođenje bezbednosti.

Konektori predstavljaju apstraktni interfejs za komunikaciju komponenti, poboljšavajući jednostavnost pružanjem jasnog razdvajanja briga i skrivanjem osnovne implementacije resursa i komunikacionih mehanizama.

Komponente REST komuniciraju tako što prenose reprezentaciju resursa u formatu koji odgovara jednom od evoluirajućeg skupa standardnih tipova podataka koji se biraju dinamički na osnovu mogućnosti ili želja primaoca i prirode resursa. Dakle, elementi podataka se mogu sumirati na sledeći način: resursi, identifikatori resursa, reprezentacije, reprezentacije i metapodaci resursa i kontrolni podaci.

3.4. REST API

Akronim API dolazi iz Application Programming Interface. API je skup funkcija i procedura koje ispunjavaju jedan ili više zadataka u svrhu da ih koristi drugi softver. Omogućava implementaciju funkcija i procedura koje su u skladu sa API-jem u drugom programu bez potrebe da se ponovo programiraju.

3.5. REST kroz HTTP

The Hypertext Transfer Protocol (HTTP) je protokol zahteva/odgovora na nivou aplikacije bez stanja koji koristi proširivu semantiku i samoopisno opterećenje poruka za fleksibilnu interakciju sa hipertekstualnim informacionim sistemima zasnovanim na mreži. U RESTful sistemu, klijenti i serveri pregovaraju o predstavljanju resursa preko HTTP-a. RESTful sistemi primenjuju četiri osnovne funkcije trajnog skladištenja, CRUD (Create, Read, Update, Delete), na skup resursa. U smislu HTTP standarda, te radnje se mogu prevesti na HTTP metode (poznate i kao glagoli): POST, GET, PUT i DELETE (Slika 2). Druge HTTP metode koje se takođe koriste, ali ne tako često kao one koje se koriste su OPTIONS, HEAD, TRACE, PATCH i CONNECT.

CRUD actions	HTTP method equivalence
Create	POST
Read	GET
Update	PUT
Delete	DELETE

Slika 2. *CRUD i HTTP ekvivalentnost*

4. JEST ALAT ZA AUTOMATIZOVANO TESTIRANJE VEB APLIKACIJA

Jest je JavaScript okvir za testiranje otvorenog koda koji je kreirao i održava Facebook. To je jedan od mnogih JavaScript okvira za testiranje, pored Mocha, Jasmine, Karme i drugih. U godišnjoj anketi o stanju JavaScript-a za 2018. u kojoj je anketirano preko 20.000 programera, Jest je pokretač testova, biblioteka tvrdnji i biblioteka za mokovanje. Jest takođe nudi ugrađeni alat za pokrivanje testa i funkciju koja se zove testiranje snimka. Za razliku od nekih svojih prethodnika, kao što je Jasmine, Jest ne zahteva pravo okruženje pretraživača za pokretanje. Umesto toga, pokreće se unutar procesa Node.js gde se API-ji pretraživača emuliraju pomoću jsdom-a.

Simulirano okruženje pretraživača ubrzava izvršavanje testova i omogućava pokretanje testova na različitim sistemima sa istim rezultatima. Isti testovi se mogu izvršiti na mašini programera i na serveru za kontinuiranu isporuku. Kao test runner, Jest pronalazi i izvršava testove. Tačnije, traži test datoteke u datom projektu i izvršava testove unutar tih datoteka. Podrazumevano, Jest proverava svaku datoteku u osnovnom direktorijumu projekta. Precizna lista fascikli za proveru može da se obezbedi kao roots svojstvo konfiguracione datoteke Jest.

Kao biblioteka tvrdnji, Jest obezbeđuje skup alata za pisanje testova na jednostavan, čoveku čitljiv način. Ovi alati su funkcije koje se nazivaju uparivači. Uparivači omogućavaju testiranje vrednosti na različite načine. Da bi se testirala određena vrednost, potrebno je kreirati objekat očekivanja. Zatim se poziva uparivač na objektu očekivanja. Kao biblioteka za mokovanje, Jest obezbeđuje alat koji se zove lažne funkcije za kreiranje testnih doubles. Lažne funkcije omogućavaju zamenu zavisnosti unutar testa jedinice nečim što se može kontrolisati i pregledati. Jest uključuje Istanbul biblioteku za

pokrivenost JavaScript koda. Istanbul obezbeđuje veliki broj reportera koji omogućavaju generisanje izveštaja u različitim formatima.

5. STUDIJA SLUČAJA - TESTIRANJE VEB APLIKACIJE ZA MANIPULACIJU KORISNIKA

Za demonstraciju automatskog RESTful API testiranja modernog softverskog projekta koristiće se jedan projekat pod nazivom „USER-API“ koji treba da pruži manipulaciju korisnika. Veb aplikacija omogaćava da se kreira korisnik, koji ima neophodne atribute kao što su ime, email i lozinka, da vrše izmene svojih atributa, kao i mogućnost brisanja. Fokus ovog poglavlja je API testiranje sa Jestom. U nastavku će biti testirane sve funkcionalnsoti koje aplikacija nudi. Aplikacija je implementirana pomoću Express radnog okvira.

Funkcionalnosti koje pokriva naša aplikaciju su kreiranje, izmenu, brisanje i dobavljanje korisnika, kao i dobavljanje svih korisnika iz baze podataka. Test scenario za brisanje korisnika je potrebno sprovesti radi utvrđivanja validnosti implementacije funkcije za brisanje korisnika. Slika 3 prikazuje funkciju za brisanje korisnika.

```
router.delete("/deleteUser/:id", (req, res, next) => {
  db.run(
    'DELETE FROM user WHERE id = ?',
    req.params.id,
    function (err, result) {
      if (err) {
        res.status(400).json({ "error": res.message });
        return;
      }
      if (this.changes !== 0) {
        res.json({ "message": "deleted", rows: this.changes, "data": result });
      } else {
        res.status(404).send("User doesn't find!");
      }
    });
});
```

Slika 3. *Prikaz funkcije za brisanje korisnika DELETE metodom*

RESTful API automatizovani test za brisanje postojećeg korisnika proverava uspešnost brisanja korisnika. Slika 4 prikazuje test za brisanje korisnika.

```
test("Should delete a user successfully", async () => {
    const EMAIL = randomEmail({ domain: 'gmail.com' });
    const requestBody = getBodyRequest(NAME, EMAIL, PASSWORD);
    const res = await request(app)
        .post(`/user/addUser`)
        .send(requestBody)
        .set('Accept', 'application/json');
    const userId = res.body.id;
    const response = await request(app)
        .delete(`/user/deleteUser/${userId}`)
        .expect('Content-Type', /json/)
        .expect(200);
    const responseForGet = await request(app)
        .get(`/user/getUser/${userId}`)
        .set('Accept', 'application/json')
        .expect('Content-Type', /json/);
    expect(responseForGet.statusCode).toBe(404);
    expect(responseForGet.body).toEqual("User doesn't find!");
});
```

Slika 4. *Prikaz testa za verifikaciju brisanja postojećeg korisnika*

6. ZAKLJUČAK

U ovom radu opisano je automatizovano testiranje u Express aplikaciji. Objašnjen je JEST radni okvir sa fokusom na RESTful API testove. Objašnjen je pojam REST. Opisani su korišćeni alati i tehnologije za razvoj aplikacije.

7. LITERATURA

[1] James Bach. Test Automation Snake Oil. http://www.satisfice. com/articles/test_automation_snake_oil.pdf, 1999. [Online; accessed 13-May-2012].

[2] Dale H. Emery. Writing Maintainable Automated Acceptance Tests. http://dhemery.com/pdf/writing_maintainable_ automated_acceptance_tests.pdf, 2009. [Online; accessed 14- May2012]

[3] M. Henning. API: Design Matters. ACM Queue, 5(4), 2018.

[4] B. A. Myers. Human-Centered Methods for Improving API Usability. In 2017 IEEE/ACM 1st International Workshop on API Usage and Evolution (WAPI), pages 2–2, May 2017.

Kratka biografija:

Aleksandar Savić rođen je u Šapcu 1998. god. Osnovne akademske studije završio je 2021. godine na Fakultetu tehničkih nauka u Novom Sadu. Master rad na Fakultetu tehničkih nauka iz oblasti Računarstvo i automatika – Računarska grafika odbranio je 2022. godine

kontakt: acasavic2409@gmail.com