Automatizirano testiranje API-ja korištenjem Cypress okruženja

Ristić, Dejana

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni diplomski studij Računarstvo

AUTOMATIZIRANO TESTIRANJE API-JA KORIŠTENJEM CYPRESS OKRUŽENJA

Diplomski rad

Dejana Ristić

Osijek, 2024.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac D1: Obrazac za ocjenu diplomskog rada na sveučilišnom diplomskom studiju

Ocjena diplomskog rada na sveučilišnom diplomskom studiju

Ime i prezime pristupnika:	Dejana Ristić
Studij, smjer:	Sveučilišni diplomski studij Računarstvo
Mat. br. pristupnika, god.	D1323R, 07.10.2022.
JMBAG:	0165079282
Mentor:	doc. dr. sc. Ivan Vidović
Sumentor:	
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	izv. prof. dr. sc. Tomislav Matić
Član Povjerenstva 1:	doc. dr. sc. Ivan Vidović
Član Povjerenstva 2:	izv. prof. dr. sc. Ivan Aleksi
Naslov diplomskog rada:	Automatizirano testiranje API-ja korištenjem Cypress okruženja
Znanstvena grana diplomskog rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak diplomskog rada:	U ovom radu potrebno je istražiti dostupne alate i okruženja za automatizirano testiranje API-ja. Nakon provedenog istraživanja potrebno je usporediti Cypress okruženje s ostalim postojećim rješenjima. U praktičnom dijelu rada potrebno je primjeniti Cypress okruženje za izradu testova na proizvoljno odabranom API-ju. Tema rezervirana za: Dejana Ristić

Datum ocjene pismenog dijela diplomskog rada od strane mentora:	20.09.2024.
Ocjena pismenog dijela diplomskog rada od strane mentora:	Izvrstan (5)
Datum obrane diplomskog rada:	8.10.2024.
Ocjena usmenog dijela diplomskog rada (obrane):	Izvrstan (5)
Ukupna ocjena diplomskog rada:	Izvrstan (5)
Datum potvrde mentora o predaji konačne verzije diplomskog rada čime je pristupnik završio sveučilišni diplomski studij:	10.10.2024.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O IZVORNOSTI RADA

Osijek, 10.10.2024.

Ime i prezime Pristupnika:	Dejana Ristić
Studij:	Sveučilišni diplomski studij Računarstvo
Mat. br. Pristupnika, godina upisa:	D1323R, 07.10.2022.
Turnitin podudaranje [%]:	6

Ovom izjavom izjavljujem da je rad pod nazivom**: Automatizirano testiranje API-ja korištenjem Cypress okruženja**

izrađen pod vodstvom mentora doc. dr. sc. Ivan Vidović

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis pristupnika:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac D1: Obrazac za ocjenu diplomskog rada na sveučilišnom diplomskom studiju

Ocjena diplomskog rada na sveučilišnom diplomskom studiju

Ime i prezime pristupnika:	Dejana Ristić
Studij, smjer:	Sveučilišni diplomski studij Računarstvo
Mat. br. pristupnika, god.	D1323R, 07.10.2022.
JMBAG:	0165079282
Mentor:	doc. dr. sc. Ivan Vidović
Sumentor:	
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	izv. prof. dr. sc. Tomislav Matić
Član Povjerenstva 1:	doc. dr. sc. Ivan Vidović
Član Povjerenstva 2:	izv. prof. dr. sc. Ivan Aleksi
Naslov diplomskog rada:	Automatizirano testiranje API-ja korištenjem Cypress okruženja
Znanstvena grana diplomskog rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak diplomskog rada:	U ovom radu potrebno je istražiti dostupne alate i okruženja za automatizirano testiranje API-ja. Nakon provedenog istraživanja potrebno je usporediti Cypress okruženje s ostalim postojećim rješenjima. U praktičnom dijelu rada potrebno je primjeniti Cypress okruženje za izradu testova na proizvoljno odabranom API-ju. Tema rezervirana za: Dejana Ristić

Datum ocjene pismenog dijela diplomskog rada od strane mentora:	20.09.2024.
Ocjena pismenog dijela diplomskog rada od strane mentora:	Izvrstan (5)
Datum obrane diplomskog rada:	8.10.2024.
Ocjena usmenog dijela diplomskog rada (obrane):	Izvrstan (5)
Ukupna ocjena diplomskog rada:	Izvrstan (5)
Datum potvrde mentora o predaji konačne verzije diplomskog rada čime je pristupnik završio sveučilišni diplomski studij:	10.10.2024.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O IZVORNOSTI RADA

Osijek, 10.10.2024.

Ime i prezime Pristupnika:	Dejana Ristić
Studij:	Sveučilišni diplomski studij Računarstvo
Mat. br. Pristupnika, godina upisa:	D1323R, 07.10.2022.
Turnitin podudaranje [%]:	6

Ovom izjavom izjavljujem da je rad pod nazivom**: Automatizirano testiranje API-ja korištenjem Cypress okruženja**

izrađen pod vodstvom mentora doc. dr. sc. Ivan Vidović

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis pristupnika:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac D1: Obrazac za ocjenu diplomskog rada na sveučilišnom diplomskom studiju

Ocjena diplomskog rada na sveučilišnom diplomskom studiju

Ime i prezime pristupnika:	Dejana Ristić
Studij, smjer:	Sveučilišni diplomski studij Računarstvo
Mat. br. pristupnika, god.	D1323R, 07.10.2022.
JMBAG:	0165079282
Mentor:	doc. dr. sc. Ivan Vidović
Sumentor:	
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	izv. prof. dr. sc. Tomislav Matić
Član Povjerenstva 1:	doc. dr. sc. Ivan Vidović
Član Povjerenstva 2:	izv. prof. dr. sc. Ivan Aleksi
Naslov diplomskog rada:	Automatizirano testiranje API-ja korištenjem Cypress okruženja
Znanstvena grana diplomskog rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak diplomskog rada:	U ovom radu potrebno je istražiti dostupne alate i okruženja za automatizirano testiranje API-ja. Nakon provedenog istraživanja potrebno je usporediti Cypress okruženje s ostalim postojećim rješenjima. U praktičnom dijelu rada potrebno je primjeniti Cypress okruženje za izradu testova na proizvoljno odabranom API-ju. Tema rezervirana za: Dejana Ristić

Datum ocjene pismenog dijela diplomskog rada od strane mentora:	20.09.2024.
Ocjena pismenog dijela diplomskog rada od strane mentora:	Izvrstan (5)
Datum obrane diplomskog rada:	8.10.2024.
Ocjena usmenog dijela diplomskog rada (obrane):	Izvrstan (5)
Ukupna ocjena diplomskog rada:	Izvrstan (5)
Datum potvrde mentora o predaji konačne verzije diplomskog rada čime je pristupnik završio sveučilišni diplomski studij:	10.10.2024.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O IZVORNOSTI RADA

Osijek, 10.10.2024.

Ime i prezime Pristupnika:	Dejana Ristić
Studij:	Sveučilišni diplomski studij Računarstvo
Mat. br. Pristupnika, godina upisa:	D1323R, 07.10.2022.
Turnitin podudaranje [%]:	6

Ovom izjavom izjavljujem da je rad pod nazivom**: Automatizirano testiranje API-ja korištenjem Cypress okruženja**

izrađen pod vodstvom mentora doc. dr. sc. Ivan Vidović

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis pristupnika:

SADRŽAJ

1. UVOD

U digitalnom svijetu, korištenje različitih vrsta softvera postalo je dio ljudske svakodnevice. Većina ljudi započinje svoj dan pogledom na neku vrstu pametnog uređaja i dobiva informacije o tome što se dogodilo na drugom dijelu svijeta, informacije o današnjem vremenu i slično. Zadovoljstvo korisnika ovisi o ispravnosti i uporabljivosti softvera, pa je softver prije isporuke potrebno testirati kako bi se utvrdila njegova ispravnost. Testiranje softvera treba započeti što ranije u ciklusu razvoja softvera jer se tako osigurava da će se greške pronaći što ranije i trošak ispravljanja greške bit će manji. Prije početka testiranja potrebno je napraviti plan testiranja prema kojem će inženjer osiguranja kvalitete provoditi testiranje softvera. Testiranje je moguće izvoditi manualno, ali je moguće i pisanje skripti za automatizirano izvođenje testova. Automatizacija omogućuje inženjerima lakše i brže provođenje testiranja. Za automatizirano testiranje postoje različiti alati, a odluka o korištenju nekoga od njih proizlazi iz mogućnosti samog alata, ali i o tome što će se testirati. Inženjeri tako mogu testirati korisničko sučelje, bazu podataka, API, ali moguće je provoditi i *e2e* (engl. *end-to-end*) testiranje koje obuhvaća sve navedeno.

U prvom poglavlju ovog diplomskog rada napravljen je uvod u temu diplomskog rada, objašnjeno je što je to testiranje i zašto je bitna stavka u razvoju kvalitetnog softvera. U drugom poglavlju predstavljen je pregled područja, koja je uloga testiranja, koje se vrste testiranja provode, a bitne su za testiranje API-ja. Opisano je što je API, na kojem principu funkcionira, formati u kojima prenosi podatke i kako je korišten u komunikaciji klijenta i servera. Također su opisani i neki od poznatijih alata za automatsko testiranje softvera. U trećem poglavlju opisan je početak pripreme za provođenje automatskog testiranje, a to je pisanje testnih slučajeva i priprema okruženja u kojem će provoditi automatizacija. U ovom poglavlju je također opisano razvojno okruženje Visual Studio Code, JavaScript okruženje Node.js, JavaScript programski jezik i Cypress alat za automatizaciju testova. U idućem, četvrtom poglavlju, opisan je API koji se testira. Prikazana je automatizacija testova kao i njihova organizacija unutar projekta. Objašnjen je način pokretanja testova i korištenje *grep tagova*. U petom poglavlju navedeni su rezultati testiranja i napravljena je usporedba izvođenja automatskog i manualnog testiranja.

2. PREGLED PODRUČJA

Potreba za testiranjem softvera postojala je oduvijek, još od stvaranja prvog programskog jezika - FORTRAN-a. Testiranje je samo jedna od faza u ciklusu razvoja softvera [1]. Uspjeh svakog softverskog rješenja leži u klijentu te se teži ispunjavanju klijentskih zahtjeva. Klijentu treba isporučiti traženi proizvod koji radi sa što manje grešaka, a testiranje osigurava upravo to. To je proces evaluiranja proizvoda koji utvrđuje ispravnost funkcionalnosti softvera, odnosno radi li softver i ponaša li se na očekivani način [2].

Postoje dvije vrste testiranja prema načinu izvođenja testova: manualno i automatsko testiranje. Manualno testiranje je testiranje koje se obavlja ručno prema testnom planu, a provjerava se funkcionalnost, upotrebljivost i softverske performanse. Pogodno je u određenim testnim situacijama, ali neke greške u softveru mogu biti zanemarene zbog ljudskog faktora. Takvo testiranje dugo traje dok kod automatskog testiranja to nije slučaj. Automatsko testiranje koristi alate za automatizaciju i skripte koje pokreću testni kôd. Ova vrsta testiranja pogodna je kod velikih projekata, kod kojih se određene funkcionalnosti često mijenjaju pa ih je potrebno i više puta testirati [3]. Ovim pristupom se taj posao ubrzava i pojednostavljuje. Automatski testovi se u većini slučajeva implementiraju u CI/CD sustav (engl. *continuous integration and continuous deployment*) koji onda svakom izmjenom kôda pokreće automatske testove [4]. Tako se provjerava ispravnost implementirane promjene i kako ta promjena utječe na ostatak kôda. Prilikom odluke o tome kako će se testirati, potrebno je procijeniti koji pristup je pogodniji. Iako manualno testiranje vremenski duže traje da bi se izvelo, sama automatizacija testova je također dugotrajan posao i zahtjeva osoblje s programerskim znanjem i vještinama. Zbog prednosti i mana oba pristupa, tijekom razvoja softvera najčešće se primjenjuje njihova kombinacija.

U ovom poglavlju opisana je teorijska podloga testiranja koja je potrebna kako bi se objasnilo automatsko testiranje API-ja.

2.1. Uloga testiranja

Unatoč mišljenju da je svrha testiranja samo pronalazak grešaka (engl. *bugs*), cilj testiranja je i osigurati da softverski proizvod radi onako kako je planirano prema njegovim zahtjevima [5]. Neke industrije troše 50% vremena na razvoj, a 50% na testiranje proizvoda [6] što dodatno ukazuje na važnost testiranja za razvoj kvalitetnog proizvoda. U prijašnjim popularnim modelima razvoja softverskog proizvoda, kao na primjer u modelu Vodopada (engl. *Waterfall model*), testiranje je zadnji korak u razvoju proizvoda. Taj pristup dovodi do velikih gubitaka

resursa koji su utrošeni na razvoj proizvoda, koji na kraju ne zadovoljava zahtjeve. Zbog kasno pronađenih pogrešaka moraju se iznova razvijati određene faze ili cijeli projekt. U danas korištenim agilnim metodologijama, testiranja proizvoda započinje u najranijoj fazi razvoja što dopušta rano otkrivanje i ispravku grešaka.

2.2. Plan testiranja

Testni plan opisuje cilj testiranja. Cilj testiranja može biti neka određena komponenta aplikacije, kao na primjer korisničko sučelje, baza podataka ili pak komunikacija korisničkog sučelja i baze, odnosno API (engl. *Application programming interface*). Prema cilju testiranja određuju se tehnologije koje su prikladne za testiranje. Također, bitno je odrediti pristup testiranja. Koja su ograničenja i na što je posebno potrebno obratiti pažnju prilikom testiranja. Važno je naglasiti da je potrebno poznavati zahtjeve koji su postavljeni prilikom plana razvoja softvera jer je to kriterij koji će biti korišten prilikom određivanja ispravnog ponašanja proizvoda. Nakon definiranja testnog plana, potrebno je napisati testne slučajeve koji će definirati korake testiranja i očekivani ishod testiranja. Testni plan predstavlja uputstva kojih se inženjer osiguranja kvalitete treba držati, kako bi testiranje bilo uspješno provedeno.

2.3. Vrste testiranja

Vrste testiranja definiraju ciljeve testiranja softverskog proizvoda [7]. Ovisno o cilju testiranja i vrsti aplikacije koja se testira, provode se različite vrste testiranja, to jest aplikacija se testira na različite načine.

2.3.1. Funkcionalno testiranje

Funkcionalnim testiranjem testira se rade li tražene funkcionalnosti softverskog proizvoda na očekivani način. Funkcionalni zahtjevi svakog softverskog proizvoda moraju biti dokumentirani u funkcionalnim specifikacijama proizvoda još i prije početka njegovog razvoja [7]. Kod funkcionalnog testiranja nije bitna pozadinska logika funkcionalnosti koja se testira. Bitno je da se za određeni unos dobije očekivani rezultat, zbog čega takvo testiranje spada u kategoriju testiranja crne kutije (engl. *black-box testing*). Općenito se funkcionalno testiranje koristi u mnogim drugim vrstama testiranja kao što je regresijsko testiranje, jedinično testiranje, integracijsko testiranje i ostalim.

2.3.2. Sigurnosno testiranje

Cilj sigurnosnog testiranja je procjena sigurnosti softverskog proizvoda [7]. Potrebno je testirati osigurava li sustav pravilnu autentifikaciju i autorizaciju korisnika, štiti li podatke korisnika, provjerava ima li sustav određenih ranjivosti, koliko je otporan na napade i slično. Ovakva vrsta testiranja iznimno je bitna jer pomaže u zaštiti sustava i korisnika, a pomaže i u otklanjanju sigurnosnih ranjivosti.

2.3.3. Unit testiranje

Unit testiranje fokusirano je na traženje grešaka u programskim jedinicama kao što su moduli, objekti i klase [7]. Kod ove vrste testiranja specifično je to da se testiranje jedinica izvodi u izolaciji od ostatka kôda, a omogućuje testiranje funkcionalnih i nefunkcionalnih karakteristika softverskog proizvoda kao i performanse, robusnost i slično.

2.4. API

API omogućuje jednostavno povezivanje, integraciju i proširivanje softverskog sustava [8]. Može se reći da API djeluje kao svojevrstan most između različitih softverskih rješenja. Omogućuje prijenos podataka u različitim formatima korištenjem protokola za razmjenu podatka.

2.4.1. API format

API formati predstavljaju način na koji se podaci razmjenjuju između klijenta i servera. Najčešće korišteni formati su JSON, XML, HTML i YAML. Svaki od ovih formata ima prednosti i nedostatke u korištenju i koriste se za različite oblike prikazivanja podataka.

Karakteristika JSON formata je jednostavnost i čitljivost i to ga čini jednim od najčešće korištenih formata API zahtjeva/odgovora [9]. Jednostavan je za parsiranje i generiranje, a često se koristi i u web aplikacijama zbog kompatibilnosti s JavaScriptom. U usporedbi s drugim formatima manje je učinkovit kada su u pitanju složene strukture podataka. Na slici 2.1. prikazan je izgled JSON formata.

```
{
   "id": 12434,
   "firstName": "John",
   "lastName": "Doe",
   "address": {
       "street": "Lakewood Blvd",
       "city": "Downey",
       "state": "CA",
       "zipCode": "90240"
   },
   "phoneNumber": "562 622-9248",
   "ssn": "1234567890"
}
```

Slika 2.1. *Prikaz JSON formata.*

2.4.2. API protokoli

Pravila i formati, koji se primjenjuju prilikom razmjene podataka putem API-ja, definirani su protokolima. Postoji nekoliko vrsta protokola, a svaki ima svoje specifične prednosti. Neki od protokola su SOAP, REST, GraphQL i gRPC. API korišten za izradu ovog diplomskog rada koristi REST protokol.

REST (engl. *Representational State Transfer*) protokol temelji se na korištenju HTTP metoda i resursa. Omogućuje komuniciranje klijenta i servera korištenjem standardnih HTTP metoda, a to su GET, POST, PUT, DELETE i druge. Prednosti ovog protokola su fleksibilnost, skalabilnost i široka podržanost.

Na slici 2.2. prikazan je API za dohvaćanje resursa sa servera. Svaki API zahtjev počinje metodom koja govori serveru šta treba napraviti sa zahtjevom, u ovom slučaju je to GET koji dohvaća resurs, i URL-a koji predstavlja putanju do tog resursa [10].

```
GET    https://parabank.parasoft.com/parabank/services_proxy/bank/customers/${userID}/accounts
```

Slika 2.2. *Prikaz korištenja GET zahtjeva.*

Funkcionalnost ranije spomenutih HTTP metoda su:

- GET: dohvaća resurs sa servera,
- POST: šalje serveru podatke koje je potrebno obraditi,

- PUT: šalje serveru zahtjev s podacima koji trebaju biti spremljeni ili ažurirani na nekom resursu,
- DELETE: briše resurs sa servera.

2.4.3. API odgovori

Neki API dokumenti mogu dati jedan odgovor u jednom kontekstu, a posve drugačiji u drugom. Kako bi se taj problem riješio i kako bi se točno znalo što se dogodilo s API zahtjevom, koriste se HTTP odgovori. Postoji čak 41 HTTP odgovor [11], a u ovom poglavlju bit će spomenuti i opisani samo neki od najpoznatijih i najčešće korištenih.

200 OK

Jedan od najčešće korištenih HTTP odgovora. Koristi se kod svih HTTP zahtjeva, a označava da je zahtjev uspješno obrađen ili da je resurs uspješno dohvaćen.

301 Moved Permanently

Status kod koji označava da je resurs koji se pokušava dohvatiti trajno premješten. To znači da je URL adresa resursa promijenjena.

400 Bad Request

Odgovor koji označava problem na klijentskoj strani zbog čega zahtjev nije uspješno obrađen. Može značiti da su problem neispravni podaci, nedostatak parametara ili problem u sintaksi samog zahtjeva.

404 Not Found

Označava da resurs nije moguće pronaći. Korišten je kada server ne zna koji resurs klijent želi dohvatiti.

401 Unauthorized

Koristi se kada klijent pristupa resursu za koji nema valjanu autorizaciju.

500 Internal Server Error

Označava problem na serverskoj strani zbog čega zahtjev nije uspješno obrađen. Vjerojatno je došlo do neočekivane greške ili iznimke.

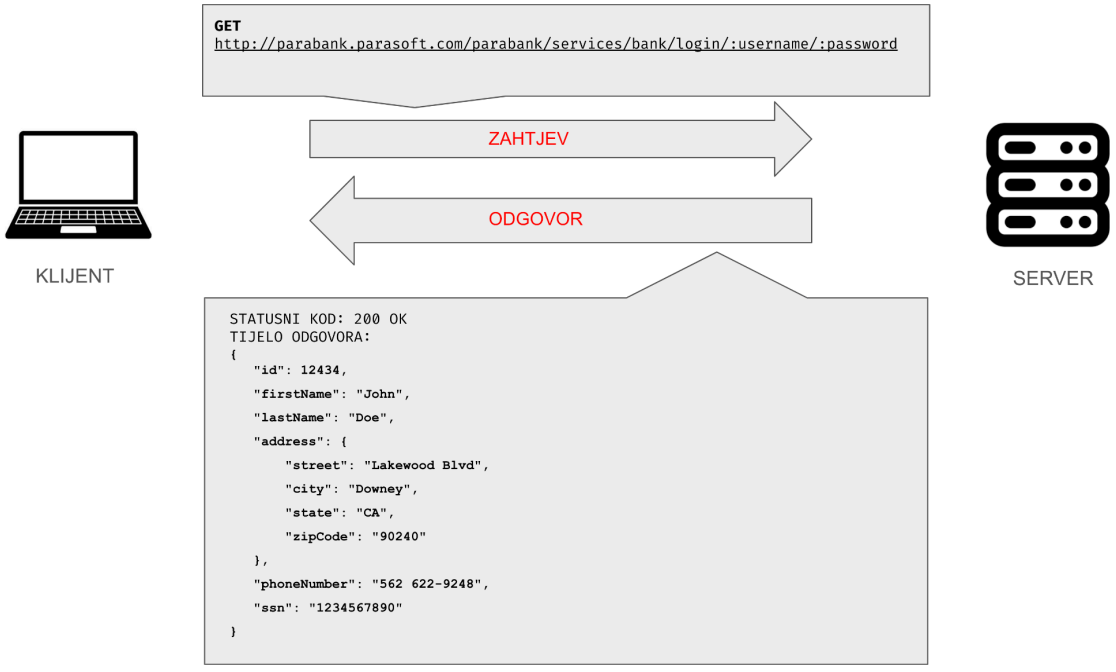
Općenito, API odgovore dijelimo u 5 skupina s obzirom na prvi broj šifre. To su:

- 1xx: informacija,

- 2xx: uspjeh,
- 3xx: preusmjeravanje,
- 4xx: pogreška na klijentskoj strani,
- 5xx: pogreška na strani servera.

2.4.4. API kao dio aplikacije

API je svojevrstan alat koji se koristi za dohvaćanje podataka putem mreže i na taj način omogućuju različitim softverima komunikaciju i razmjenu podataka. Prilikom korištenja, API nije pozvan direktno od strane klijenta već ga poziva softver koji klijent koristi. Na slici 2.3. prikazan je princip rada API-ja. S jedne strane je aplikacija koju koristi krajnji korisnik. To može biti mobilna aplikacija, web aplikacija ili bilo koja druga vrsta softvera. U ovom slučaju to se naziva klijentom. S druge strane, nalazi se server sa resursima kojima se želi pristupiti. Razmjena poruka između klijenta i servera podrazumijeva slanje zahtjeva od klijenta k serveru i primanje odgovora od strane servera. Zahtjev se šalje korištenjem HTTP metoda spomenutih u ranijem poglavlju i URL-a koji predstavlja putanju do resursa koji se želi dohvatiti. Nakon što server primi zahtjev, zahtjev se obrađuje. Ovisno o tome kako je server obradio zahtjev, klijentu se vraća statusni kôd i tijelo odgovora [10].



Slika 2.3. *Prikaz komunikacije klijenta i servera.*

2.5. Alati za automatsko testiranje

Kako bi počelo automatsko testiranje, potrebno je odabrati prikladan alat za automatizaciju testova. Prema [12], pri odabiru alata za testiranje potrebno je uzeti u obzir sljedeće:

- vrsta aplikacije koja se testira (web, mobilna, desktop),

- vrsta testiranja koja se provodi (regresijsko testiranje, unit testiranje, testiranje performansi),

- analiza cijelog testnog scenarija,

- trošak obuke zaposlenika,

- trošak alata za automatsko testiranje.

Na tržištu postoje različite vrste komercijalnih i besplatnih alata kojima se može provoditi automatsko testiranje. Neki od poznatih su Appium, TestNG, Apache JMeter, Cypress, Postman, Selenium. U ovom poglavlju bit će opisani neki od najpoznatijih alata za automatsko testiranje.

2.5.1. Appium

Appium je alat korišten za automatizirano UI testiranje iOS, Android i Windows aplikacija, a omogućuje testiranje mobilnih i hibridnih aplikacija. Java, JavaScript, C#, Ruby i Python programski jezici mogu biti korišteni za pisanje skripti za automatizirano pokretanje testova. Arhitektura Appiuma se sastoji od programskog kôda koji prilikom izvođenja pokreće automatizirane skripte na Android ili iOS platformi. Appium Inspector je dodatan alat koji omogućuje laku interakciju s elementima aplikacije. Mana mu je to što može biti sporiji u odnosu na neke druge alate za automatizirano testiranje [14].

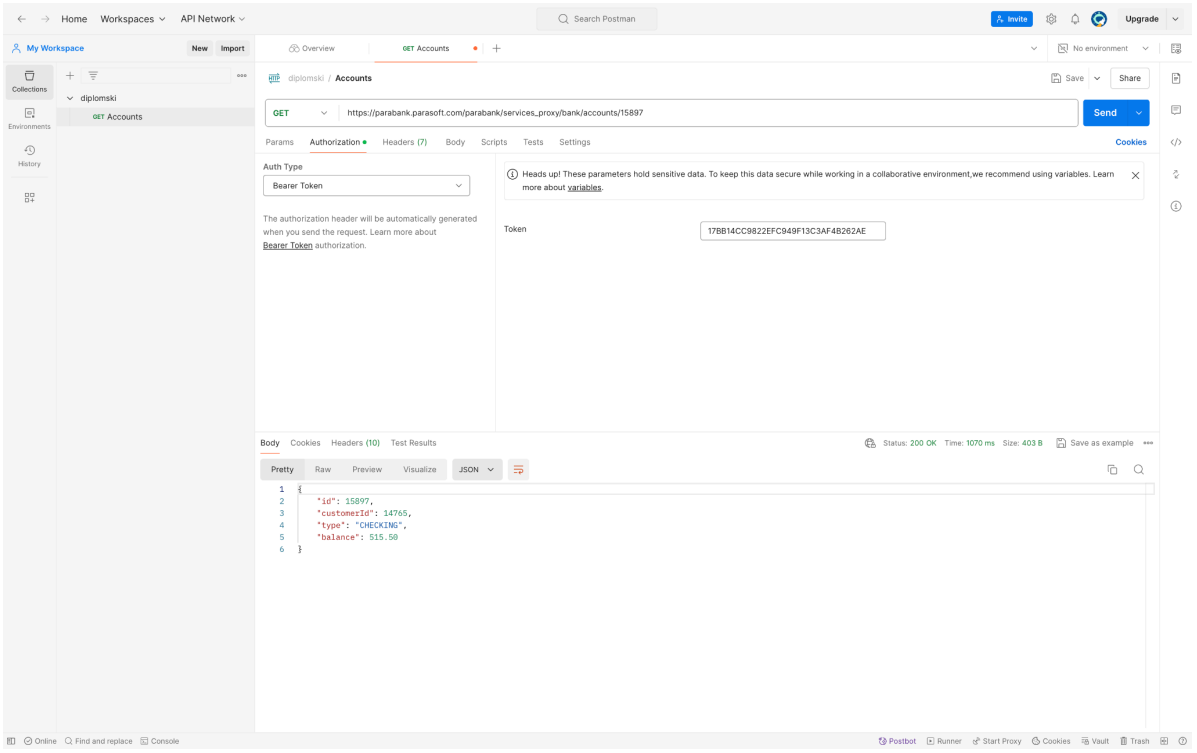
2.5.2. Selenium

Selenium je *open source* razvojni okvir (engl. *framework*) korišten za automatizaciju testova. Omogućuje pokretanje testova u bilo kojem web pregledniku. Sadrži Selenium WebDriver komponentu koja dopušta nativne interakcije s web preglednikom kao što su klik, slanje podataka, popunjavanje formi i slično. Popularan je zbog svoje robusnosti, fleksibilnosti i podrške za različite tehnologije i programske jezike [15].

2.5.3. Postman

Postman je alat koji omogućuje lako testiranje i debugiranje API-ja. Podržava i slanje kompleksnih zahtjeva, pisanje testnih skripti kao i organiziranje API-ja u kolekcije. Testovi

napisani u JavaScript-u mogu se pokretati ručno, ali se mogu implementirati i u CI/CD *pipeline*-u [16]. Na slici 2.3. prikazan je izgled Postman sučelja.



Slika 2.3. *Sučelje Postman-a.*

2.5.4. Cypress

Cypress je popularan alat za automatizirano testiranje korištenjem programskog jezika JavaScript. Najčešće je korišten za frontend testiranje web aplikacija, ali može se koristiti za testiranje svega što se pokreće u pretraživaču. Omogućuje postavljanje, pisanje, izvođenje i debugiranje testova na brži, lakši i pouzdaniji način [13].

3. AUTOMATIZIRANO TESTIRANJE API-JA

S obzirom na to da se testiranje softvera smatra jednako bitnim kao i sam razvoj, na proces testiranja je potrebno uložiti puno vremena. Inženjeri su pokušali doskočiti tom problemu uvođenjem automatizacije izvođenja testova. Automatizacija omogućuje korištenje različitih programskih alata za pisanje automatiziranih skripti za izvođenje testova. Tako ne samo da se smanjuje vrijeme izvođenja testova, nego se povećava i njihova točnost. Inženjeri kvalitete ne moraju svaki put izvoditi iste radnje iznova što smanjuje umor i mogućnost greške. Također, skripte za automatizirano testiranje moguće je po potrebi ažurirati ako su se određeni zahtjevi na softver mijenjali, ali ih je moguće i reciklirati za potrebe testiranja nekog drugog softvera.

Automatizacija testiranja bilo kojeg softvera, pa tako i API-ja, najprije započinje definiranjem testnih slučajeva i odabira alata za automatizaciju. Tijek automatiziranog testiranja API-ja korištenog za izradu ovog diplomskog rada bit će opisan u narednim poglavljima.

3.1. Priprema testnih slučajeva

Testni slučajevi su svojevrstan plan testiranja u kojima se detaljno opisuje koji dio softvera će se testirati i na koji način, a nastaju prilikom definiranja specifikacija na softver. U testnim slučajevima dani su točni koraci koje je potrebno poduzeti kako bi se testiranje uspješno provelo, kao i očekivani rezultat po završetku izvođenja. Prilikom planiranja testiranja posebna se pažnja obraća na alate i tehnologije korištene u razvoju softvera koji se testira. Svaki od njih ima svoje specifičnosti koje mogu utjecati na softver, ali i na rezultate testiranja [17]. Testni slučajevi moraju pokriti određeni dio softvera (engl. *code coverage*). Pokrivenost kôda je mjera koja opisuje koji dio kôda je pokriven testiranjem i koliko je kôd temeljito testiran. Što je veći postotak pokrivenosti kôda testiranjem, to je veća i vjerojatnost da će softver raditi ispravno, jer je veća mogućnost pronalazaka grešaka u kôdu i njihova ispravljanja. Testiranja je potrebno započeti što ranije, odnosno čim je napisan kôd za funkcionalnost koja se testira. Tako se smanjuje vrijeme koje je potrebno za testiranje na kraju razvoja softvera.

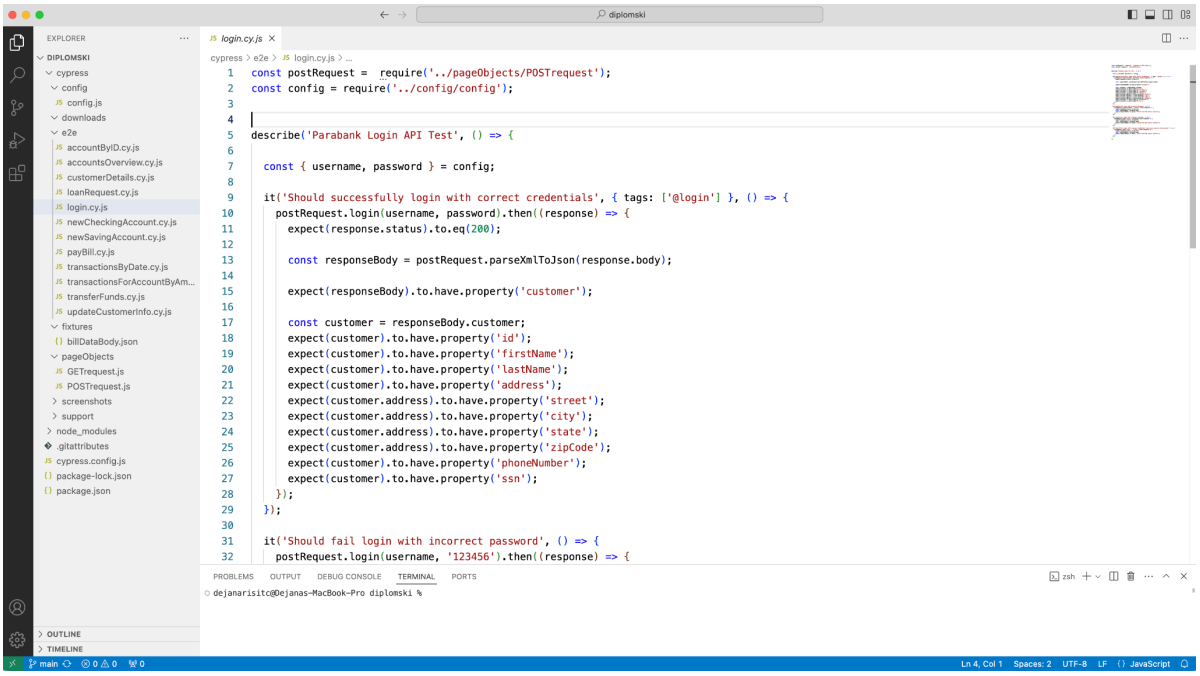
3.2. Priprema testnog okruženja

Prije početka automatizacije testiranja potrebno je odabrati uređivač kôda u kojem će se kôd pisati. U izradi ovog diplomskog rada korišten je Visual Studio Code. Testne skripte su pisane u programskom jeziku JavaScript, u alatu za automatizirano testiranje - Cypress. Svaka od ovih stavki se može razlikovati i njihov odabir ovisi o softveru koji se testira, ali i o znanjima i vještinama osoba koje provode testiranje.

3.2.1. Visual Studio Code

Visual Studio Code je Microsoftov *open-source* uređivač kôda. Idealan je za razvoj aplikacija na različitim programskim jezicima, uključujući JavaScript, Python, C++, Java i mnoge druge. Visual Studio Code je brzo i prilagodljivo programsko okruženje, a bogat je i mnogobrojnim ekstenzijama koje omogućavaju brži i lakši razvoj kôda.

Na slici 3.1. vidljivo je da Visual Studio Code omogućuje lako i intuitivno korištenje. S lijeve strane nalazi se prikaz strukture direktorija. U gornjem dijelu prozora nalazi se polje za pretraživanje, koje omogućuje pretraživanje datoteka prema nazivu ili određenom tekstu. Omogućuje integraciju s drugim alatima za kontrolu verzija kôda (engl. *Source Control*), kao što je Git. Sadrži alate za debugiranje i pokretanje aplikacija [18]. Sve ovo čini Visual Studio Code jednim od popularnijih alata korištenih u razvoju kôda.



Slika 3.1. *Sučelje Visual Studio Code-a.*

3.2.2. Node.js

Node.js je cross-platformsko, runtime okruženje koje omogućuje izvođenje JavaScript kôda izvan preglednika [19]. U sklopu Node.js-a razvijen je i NPM (engl. *Node Package Manager*) koji omogućuje upravljanje JavaScript i Node.js paketima. Olakšava instalaciju, ažuriranje, upravljanje i dijeljenje biblioteka i alata koji se koriste prilikom razvoja. Korišten je i za instalaciju Node.js-a i Cypress-a [20] potrebnih u izradi ovog diplomskog rada.

3.2.3. JavaScript

JavaScript je objektno-orijentirani programski jezik. Omogućuje manipulaciju DOM-om (engl. *Document Object Model*) pa je zbog toga popularan za korištenje prilikom izrade web aplikacija. Također, omogućuje reakciju na korisničku interakciju poput klika miša, unosa s tipkovnice i slično. Postoji nekoliko poznatih JavaScript biblioteka i razvojnih okruženja prigodnih za razvoj testnih skripti.

U ovom diplomskom radu korištena je Chai biblioteka za provjeru očekivanih ishoda (engl. *assertion*) [21]. Ova biblioteka nudi tri različite sučelja za provjeru, a to su *should*, *expect* i *assert*. Na slici 3.2. prikazan je načina korištenja *expect* sučelja.

```
it("Fetch transactions for specific date for account", () => {

    getRequest.getRequest(apiUrl2).then((response) => {
        expect(response.status).to.eq(200);
        const transfer = response.body[0];
        expect(transfer).to.have.property('id');
        expect(transfer).to.have.property('accountId');
        expect(transfer).to.have.property('date');
        expect(transfer).to.have.property('amount');
        expect(transfer).to.have.property('description');
    });

});
```

Slika 3.2. *Implementacija expect sučelja Chai biblioteke.*

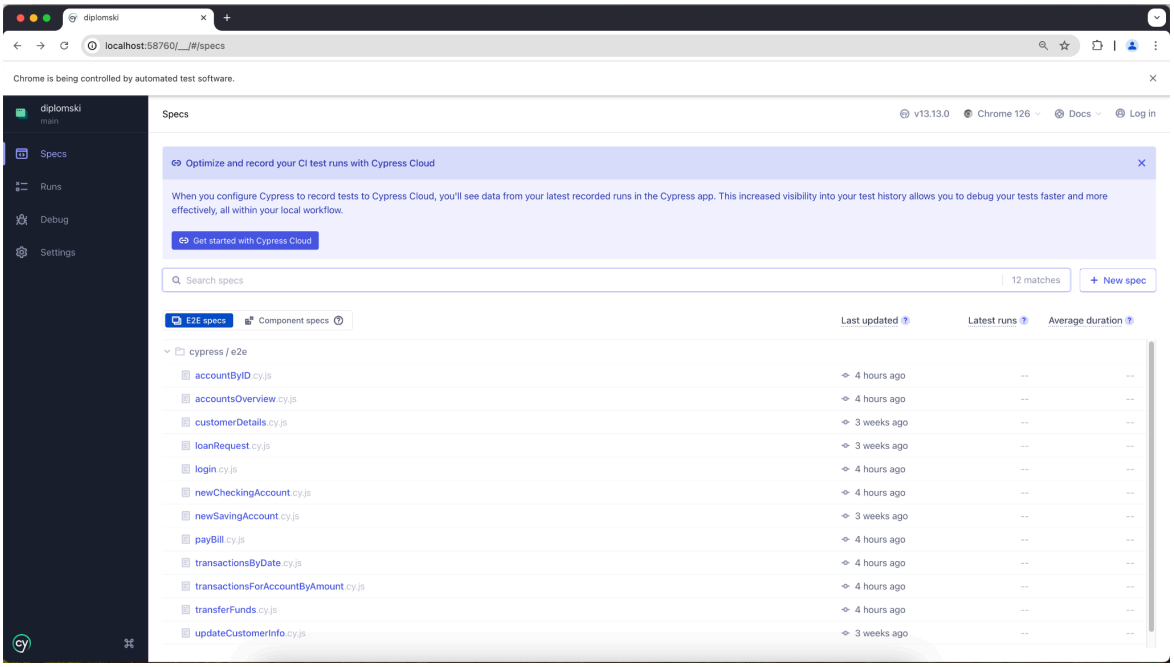
Mocha testno okruženje omogućava asinkrono izvođenje testova i može raditi s bilo kojom bibliotekom za provjeru ishoda [22]. Pogodna je zbog korištenja takozvanih *hook*-ova koji postavljaju uvjete koji se trebaju izvesti prije ili nakon izvođenja seta testova. *Hook before()* izvodi se jednom prije početka svih testova u datoteci, a *hook* after() izvodi se jednom na kraju svih testova u datoteci. *Hook beforeEach()* izvodi se prije izvođenja svakog testa iz datoteke, a *afterEach()* izvodi se nakon izvođenja svakog testa iz datoteke. Slika 3.3. prikazuje formu svakog *hook*-a i njihovu funkcionalnost.

```
describe('hooks', function () {
  before(function () {
   // pokrece se jednom prije prvog testa u bloku
  });

  after(function () {
    // pokrece se jednom nakon zadnjeg testa u bloku
  });

  beforeEach(function () {
    // pokrece se prije svakog testa u bloku
  });

  afterEach(function () {
    // pokrece se nakon svakog testa u bloku
  });

  // testni slucajevi
});
```

Slika 3.3. *Mocha hook-ovi i njihova implementacija.*

3.2.4. Cypress

Kao što je u prethodnom poglavlju spomenuto, Cypress je alat korišten za automatizaciju testova korištenjem JavaScript programskog jezika. Može testirati sve što se pokreće u pregledniku. Izbornik prikazuje sve testove koji se nalaze u datoteci iz koje je pokrenut Cypress. Izgled izbornika prikazan je na slici 3.4., a tu je moguće odabrati koja testna skripta će se pokrenuti.



Slika 3.4. *Izgled Cypress sučelja.*

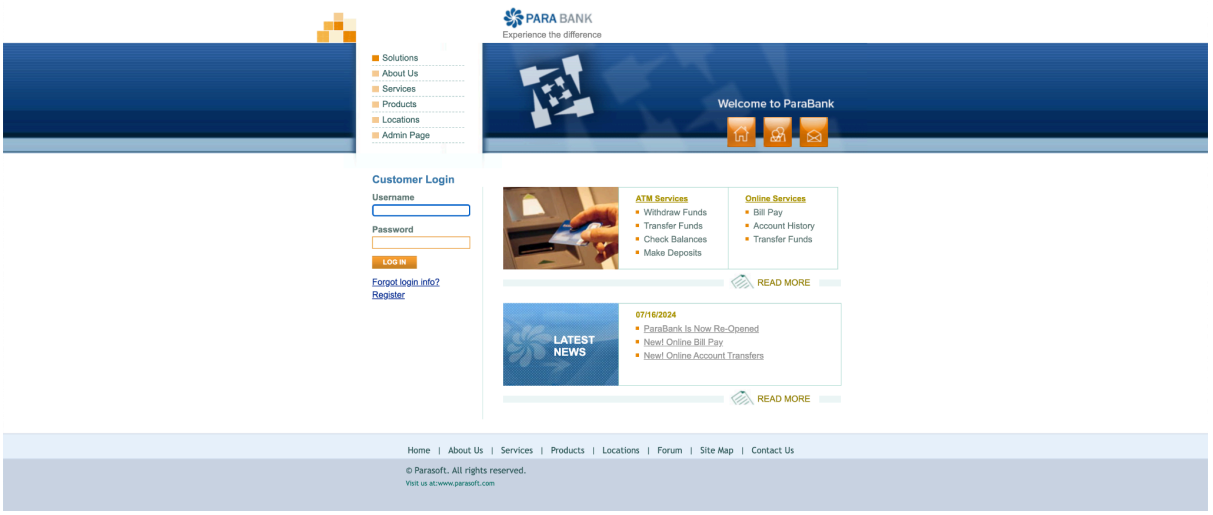
U poglavlju 4.2.2. objašnjeno je pokretanje testne skripte unutar terminala u Visual Studio Code-u.

4. IMPLEMENTACIJA TESTOVA

U ovom poglavlju opisan je proces testiranja ParaBank testnog API-ja, priprema testnog okruženja i implementacija testova u Cypress-u. Također, opisana je struktura testova i njihova organizacija korištenjem modela objekta stranice (engl. *Page Object Model - POM*).

4.1. ParaBank REST API

ParaBank REST API je testni API tvrtke Parasoft. API simulira funkcije internet bankarstva, a vezan je za njihovu demo web stranicu (Slika 4.1.). Dokumentaciji ParaBank API-ja može se pristupiti putem poveznice: https://parabank.parasoft.com/parabank/api-docs/index.html.



Slika 4.1. *Izgled sučelja ParaBank web aplikacije.*

U ovom diplomskom radu testirane su funkcionalnosti:

- prijava korisnika,
- pregled korisničkih računa,
- pregled detalja korisničkog računa,
- dohvaćanje podataka o korisniku,
- zahtjev za pozajmicom,
- otvaranje tekućeg računa,
- otvaranje štednog računa,
- plaćanja,
- filtriranje transakcija prema datumu,
- filtriranje transakcija prema iznosu,
- prijenos novca,

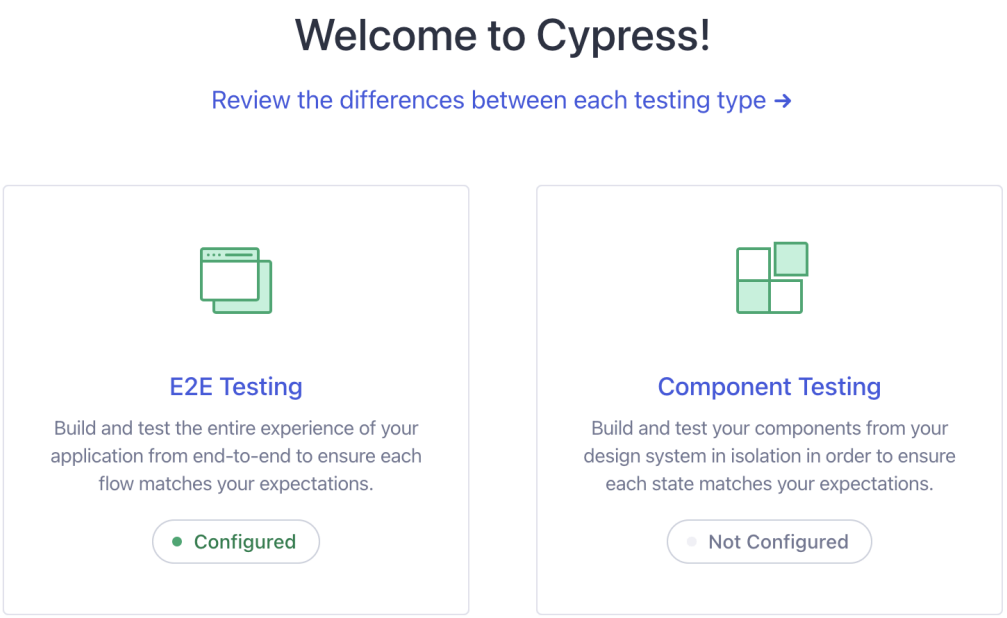
- ažuriranje korisničkih podataka.

API sadrži i *endpointove* za administratorske funkcionalnosti koje je također moguće testirati, ali taj dio izlazi iz okvira ovog diplomskog rada. Testni slučajevi prema kojima je API testiran nalaze se u Prilogu 2.

4.2. Cypress okruženje

Kako bi se testovi mogli pokrenuti, potrebno je unutar projekta instalirati Cypress. Prije toga, potrebno je instalirati Node.js s poveznice na njihovu službenu stranicu: https://nodejs.org/en/download/package-manager. Nakon što je instalacija završena, pokreće se instalacija Cypress-a korištenjem naredbe `npm install cypress --save-dev` koja se unosi u terminal unutar Visual Studio Code-a.

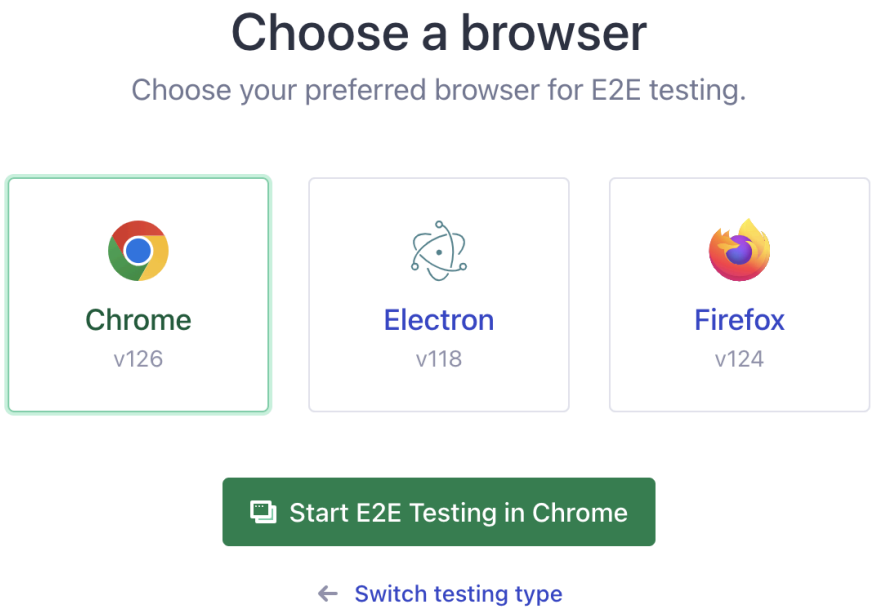
Ako se testovi žele pokrenuti unutar pretraživača, Cypress se pokreće naredbom `npx cypress open`. Na slici 4.2. prikazan je izgled sučelja u kojem se odabire koja vrsta testova će se pokretati i potrebno je odabrati *E2E Testing* opciju.



Slika 4.2. *Sučelje Cypress-a.*

Zatim se otvara sučelje za izbor preglednika u kojem će se testovi pokretati. U izradi ovog diplomskog rada korišten je *Chrome* preglednik (Slika 4.3.). Izbor preglednika definiran je u testnom planu. Ponekad je potrebno softver testirati i na nekoliko različitih preglednika. Moguće

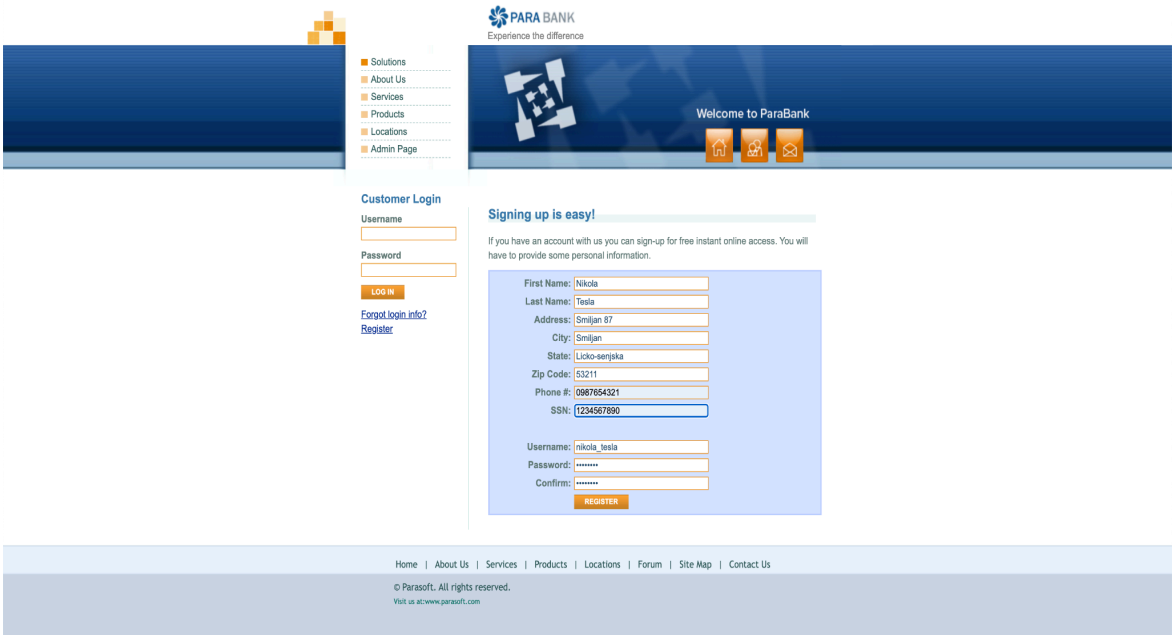
je postojanje ograničenja zbog kojih bi ponašanja na različitim preglednicima mogla biti drugačija od očekivanih te je zbog toga potrebno dodatno testiranje. Klikom na *Start E2E Testing in Chrome* opcije, otvara se sučelje sa svim testovima koji se nalaze u projektu.



Slika 4.3. *Sučelje odabira pretraživača.*

4.2.1. Struktura testova unutar Cypress-a

S obzirom na to da razvojni inženjeri API-ja u dokumentaciji nisu definirali *endpoint* za registraciju korisnika, registracija nije automatizirana i potrebno ju je izvršiti na web stranici ParaBank-a. Na slici 4.4. prikazana je forma za registraciju.



Slika 4.4. *Sučelje za registraciju korisnika na ParaBank web aplikaciji.*

Nakon uspješne registracije, u *config.js* dokumentu potrebno je izmijeniti *username* i *password* korisnika. Ovi podaci potrebni su za uspješno izvođenje testova. Na slici 4.5. prikazana je *config.js* datoteka.

config.js

cypress > config > config.js > ...

```
1  //register on: https://parabank.parasoft.com/parabank/register.htm
2  //after successful registration, enter username and password to successfully run tests
3  module.exports = {
4      username: 'nikola_tesla',
5      password: '12345678',
6  };
7
```

Slika 4.5. *Config.js dokument.*

Testne skripte (engl. *spec file*) u Cypress-u nalaze se u *cypress/e2e* datoteci [23]. Sve testne skripte moraju imati ekstenziju *.cy* koja označava da je riječ o Cypress testovima. Na početku skripte nalaze se linije kôda koje uvoze (engl. *import*) druge datoteke i omogućuju korištenje funkcionalnosti koje se nalaze u njima. Nakon toga nalazi se *describe()* metoda koja za argument prima parametar tipa *string* koji opisuje testove koji se nalaze unutar bloka. Svaki test započinje *it()* metodom. Unutar jednog *describe()* bloka može se nalaziti više *it()* blokova testova koji se ponašaju kao zasebni testovi, odnosno neovisni su od ostalih testova u bloku. Slika 4.6. prikazuje skriptu koja testira API zahtjev za plaćanje računa. Također, ovdje je prikazano i korištenje Mocha *before() hook*-a koji omogućuje izvođenje kôda prije svih testova u bloku. Unutar *it()* bloka nalazi se *expect* sučelje *Chai* biblioteke koja provjerava je li dobiveni rezultat jednak

očekivanom. Tako je moguće provjeriti je li API vratio očekivani string, ima li povratni podatak očekivano svojstvo (engl. *property*) i slično.

```
const getRequest =  require('../pageObjects/GETrequest');
const postRequest =  require('../pageObjects/POSTrequest');
const apiNonStatic = require('../pageObjects/ApiNonStatic');
const config = require('../config/config');

describe('Parabank Pay Bill API Test', () => {
    const { username, password } = config;
    let userID;
    let apiUrl1;
    let accountID;
    const invalidID = apiNonStatic.generateRandomString(10);

    before('Should successfully login with correct credentials', () => {

        postRequest.login(username, password).then((response) => {
            expect(response.status).to.eq(200);
            const responseBody = postRequest.parseXmlToJson(response.body);
            expect(responseBody).to.have.property('customer');
            const customer = responseBody.customer;
            userID = customer.id._text.replace(/"/g, '');
            apiUrl1 = `https://parabank.parasoft.com/parabank/services_proxy/bank/customers/${userID}/accounts`;
          });

    });

    it("Get all bank accounts of the user by valid user ID", () => {
        getRequest.getRequest(apiUrl1).then((response) => {
            expect(response.status).to.eq(200);
            const account = response.body[0];
            expect(account).to.have.property('id');
            accountID = account.id;
            apiUrl1 =  `https://parabank.parasoft.com/parabank/services_proxy/bank/billpay?accountId=${accountID}&amount=200`;
        });
    });
```

Slika 4.6. *Skripta za dohvaćanje bankovnih računa korisnika.*

Fixture datoteke sadrže unaprijed definirane podatke koji se mogu koristiti u testovima. U *fixture* datoteci ovog projekta nalazi se dokument s podacima u JSON formatu (Slika 4.7.). Ovaj podatak koristi se kao argument koji se predaje POST funkciji kao dio tijela (engl. *body*) zahtjeva koji se šalje serveru. U ovom slučaju, to su podaci potrebni za izvršavanje plaćanja.

```
{
  "billDataBody": [
    {
     "name": "User",
      "address": {
          "street": "Userly",
          "city": "Usertown",
          "state": "Osjecko-baranjska",
          "zipCode": "31000"
      },
      "phoneNumber": "0987654321",
      "accountNumber": 12345
    }
  ]
}
```

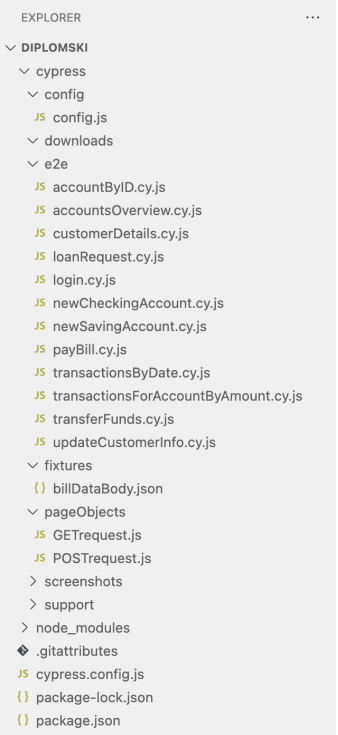
Slika 4.7. *Podaci iz* Fixtures *datoteke.*

Fixture podaci se u testu predaju kao argument *cy.fixture()* metode [24]. Na slici 4.8. prikazana je sintaksa.

```
it("Pay bill with valid accountId and amount using valid body data", () => {
    cy.fixture('billDataBody').then((data) => {
        postRequest.postRequestWithBodyData(apiUrl1, data.billDataBody[0]).then((response) => {
            expect(response.status).to.eq(200);
            const account = response.body;
            expect(account).to.have.property('payeeName');
            expect(account).to.have.property('amount');
            expect(account).to.have.property('accountId');
        });
    });

});
```

Slika 4.8. *Sintaksa cy.fixture() metode.*

Za organizaciju kôda korišten je model objekta stranice. Iako je POM asociran s testiranjem korisničkog sučelja, za organizaciju i upravljanje web elementima, taj se princip može koristiti i u testiranju API-ja radi poboljšanja organizacije, preglednosti i olakšavanja održavanja kôda. Na slici 4.9. prikazana je struktura projekta. Svaki API zahtjev definiran je kao klasa, a ta klasa sadrži metode koje se pozivaju u testovima. Slika 4.10. prikazuje klasu GET zahtjeva.



Slika 4.9. *Struktura projekta.*

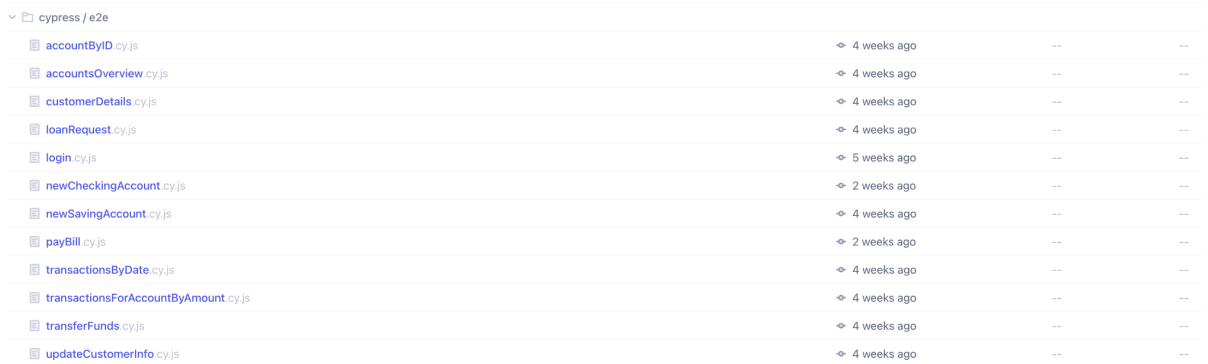
```
class GetRequest {

  getRequest(apiUrl) {
    return cy.request({
        method: 'GET',
        url: apiUrl,
        failOnStatusCode: false,
        headers: {
            'Content-Type': 'application/json',
            'Authorization': 'Bearer '
        }
    })
  }
  getRequestNoAutorization(apiUrl) {
    return cy.request({
        method: 'GET',
        url: apiUrl,
        failOnStatusCode: false,
        headers: {
            'Content-Type': 'application/json'
        }
    })
  }
}

module.exports = new GetRequest();
```

Slika 4.10. *Klasa GET zahtjeva.*

4.2.2. Pokretanje testova

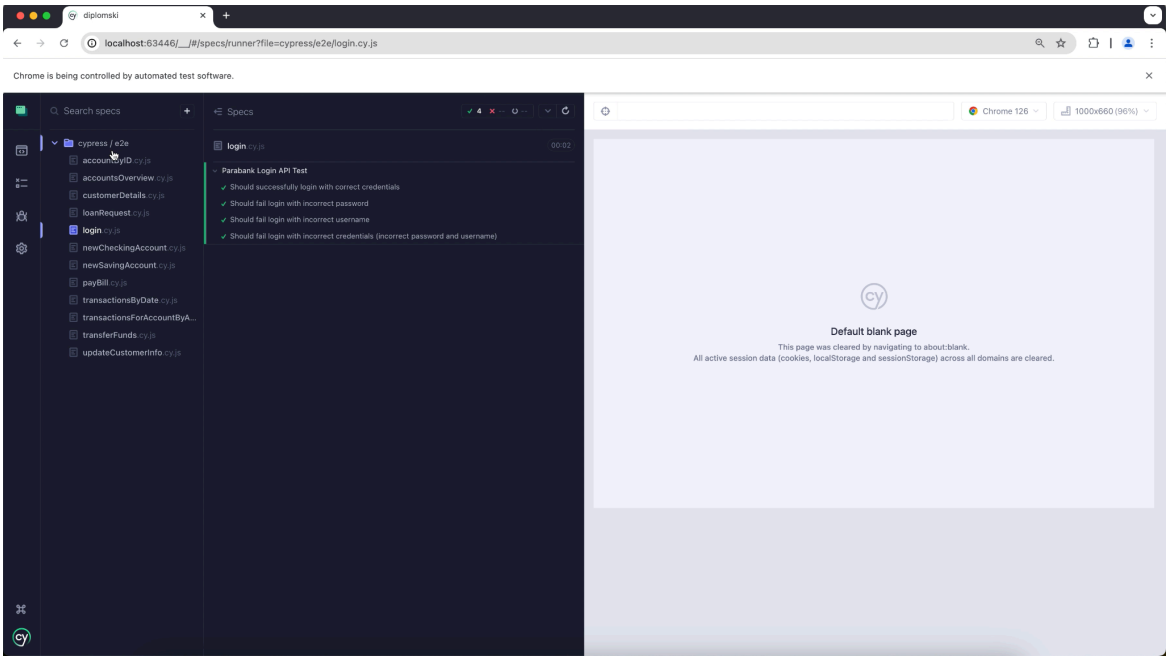
Da bi se testne skripte izvele, potrebno ih je najprije pokrenuti. Cypress omogućuje dva načina pokretanja testova, pokretanje u pregledniku i pokretanje u terminalu (engl. *headless*).

Testovi se u pregledniku pokreću unošenjem naredbe `npx cypress open` u terminal. Nakon što se Cypress pokrenuo, potrebno je odabrati vrstu testiranja, a to je *E2E Testing,* a zatim preglednik u kojem se testira - *Chrome*. Na slici 4.11. prikazane su sve testne skripte koje se nalaze u projektu, odnosno u *e2e* datoteci.



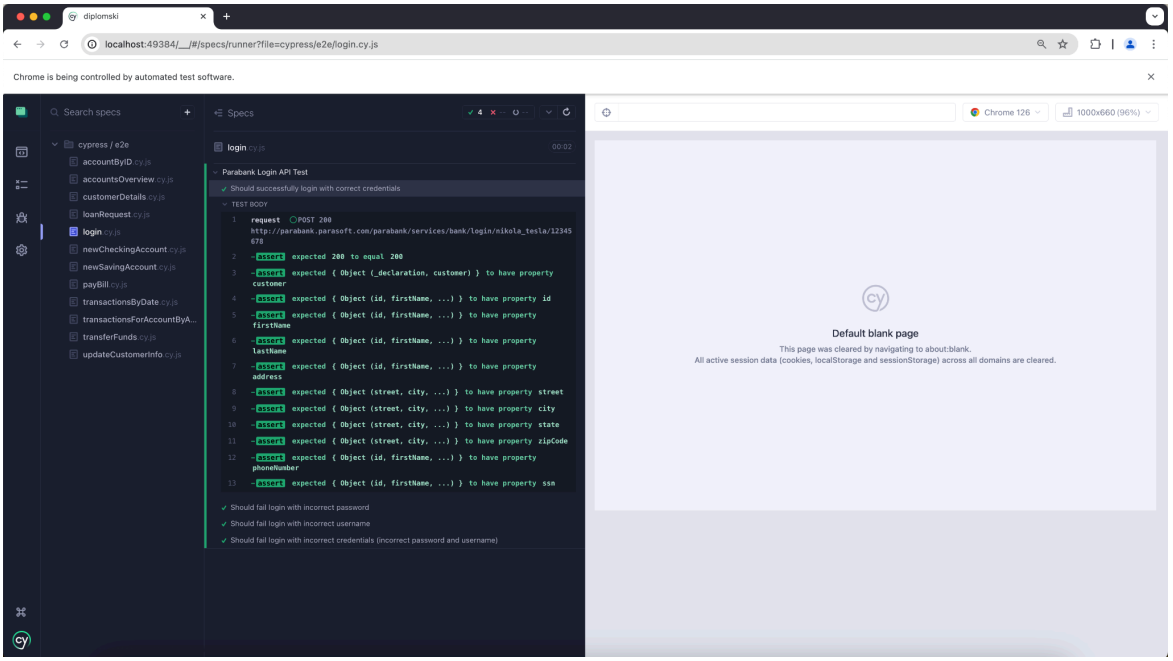
Slika 4.11. *Prikaz testnih datoteka u Cypress-u.*

Nakon odabira testne skripte, ta skripta započinje sa izvođenjem. Na slici 4.12. s lijeve strane prikazani su svi izvedeni testovi unutar skripte, a s desne se u slučaju testiranja korisničkog sučelja nalazi vizualni prikaz elemenata koji se testiraju.



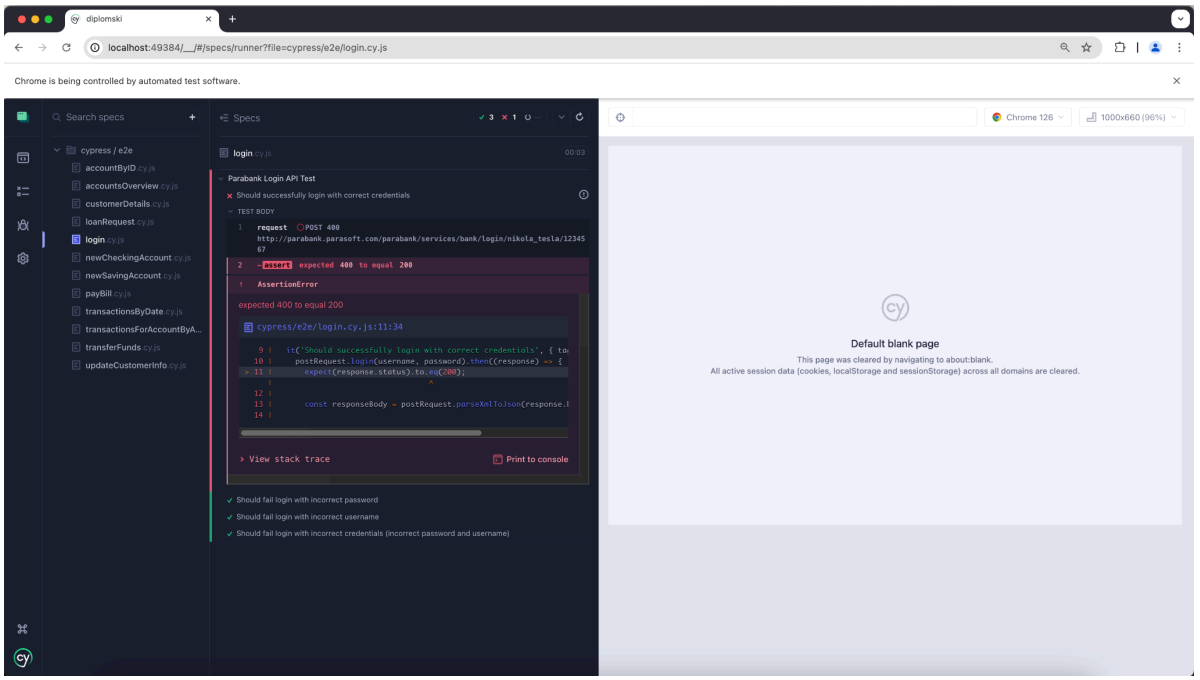
Slika 4.12. *Izvođenje testne skripte u Cypress-u.*

Na slici 4.13. vidljivo je da se klikom na test prikazuju koraci izvođenja testova. Ti koraci mogu služiti kao pomoć prilikom pronalaska grešaka u kôdu.



Slika 4.13. *Prikaz koraka izvršavanja testa.*

Testovi koji prolaze označeni su zelenom bojom teksta, a oni koji padaju crvenom. Na slici 4.14. nalazi se primjer testa koji nije uspješno izveden.



Slika 4.14. *Prikaz neuspješno izvedenog testa.*

Drugi način pokretanja je *headless*, odnosno izvođenje testova u terminalu. Testovi se pokreću unošenjem naredbe `npx cypress run` u terminal. Na slici 4.15. prikazan je tijek izvođenja *login.cy.js* skripte unutar terminala. Testovi se u terminalu također izvode neovisno kao i u pretraživaču. Na kraju izvođenja skripte prikazana je tablica s rezultatima izvođenja.

```
Running:  login.cy.js                                                    (5 of 12)

Parabank Login API Test
  ✓ Should successfully login with correct credentials (361ms)
  ✓ Should fail login with incorrect password (578ms)
  ✓ Should fail login with incorrect username (302ms)
  ✓ Should fail login with incorrect credentials (incorrect password and username) (316ms)

4 passing (2s)

(Results)

┌──────────────┬──────────────┐
│ Tests:       │ 4            │
│ Passing:     │ 4            │
│ Failing:     │ 0            │
│ Pending:     │ 0            │
│ Skipped:     │ 0            │
│ Screenshots: │ 0            │
│ Video:       │ false        │
│ Duration:    │ 1 second     │
│ Spec Ran:    │ login.cy.js  │
└──────────────┴──────────────┘
```

Slika 4.15. *Rezultat pokretanja login.cy.js skripte u terminalu.*

U slučaju da test nije uspješno izveden, odnosno rezultat nije jednak očekivanom, test je označen crvenom bojom teksta. Također, Cypress će u terminalu ispisati i razlog pada tog testa, kao i

tablicu konačnih rezultata testiranja. Na kraju prikazuje i snimak zaslona pada testa koji je naročito koristan u slučaju testiranja korisničkog sučelja. Slika 4.16. prikazuje neuspješno izveden test.

```
Running:  login.cy.js                                                    (5 of 12)


Parabank Login API Test
  1) Should successfully login with correct credentials
  ✓ Should fail login with incorrect password (341ms)
  ✓ Should fail login with incorrect username (318ms)
  ✓ Should fail login with incorrect credentials (incorrect password and username) (312ms)


3 passing (2s)
1 failing

1) Parabank Login API Test
     Should successfully login with correct credentials:

    AssertionError: expected 400 to equal 200
    + expected - actual

    -400
    +200

    at Context.eval (webpack:///./cypress/e2e/login.cy.js:11:33)



(Results)

┌────────────────────────────────────────────────────────────────────────────┐
│ Tests:        4                                                            │
│ Passing:      3                                                            │
│ Failing:      1                                                            │
│ Pending:      0                                                            │
│ Skipped:      0                                                            │
│ Screenshots:  1                                                            │
│ Video:        false                                                        │
│ Duration:     1 second                                                     │
│ Spec Ran:     login.cy.js                                                  │
└────────────────────────────────────────────────────────────────────────────┘


(Screenshots)

-  /Users/dejanarisitc/Documents/diplomski/cypress/screenshots/login.cy.js/Parabank    (2560x1440)
    Login API Test -- Should successfully login with correct credentials (failed).p
   ng
```

Slika 4.16. *Prikaz pada testa u terminalu.*

4.2.3. Grep tags

U velikim projektima koji sadrže mnogobrojne testne skripte je ponekad potrebno dodatno filtrirati testove umjesto da se pokreću cijele testne skripte. U svrhu filtriranja mogu se koristiti *grep tag-ovi*. Oni predstavljaju dodatnu informaciju koja se može koristiti za filtriranje. *Grep* se instalira naredbom `npm i -D @cypress/grep`. U *cypress.config.js* datoteku dodaje se kôd sa slike 4.17. koji je dostupan na poveznici: https://www.npmjs.com/package/@cypress/grep.

```
const { defineConfig } = require('cypress')
module.exports = defineConfig({
  e2e:
  {
    setupNodeEvents(on, config) {
      require('@cypress/grep/src/plugin')(config);
      return config;
    }
  }
})
```

Slika 4.17. *Slika kôda za postavljanje grep tag-ova u projektu.*

Grep tag-ovi dodaju se kao argument *it()* funkcije. Na slici 4.18. prikazano je dodavanje *@login tag-a* u test.

```
it('Login with correct credentials', { tags: ['@login'] }, () => {
```

Slika 4.18. *Dodavanje @login grep tag-a u test.*

Grep omogućuje nekoliko načina filtriranja testova. Njegovim korištenjem moguće je pokrenuti sve testove koji ne sadrže *tag*, testove koji sadrže određenu riječ u opisu testa, kombinaciju nekoliko *grep tag-ova* i slično. Za pokretanje testova označenih s *@login tag-om*, u terminal je potrebno unijeti naredbu `npx cypress run --env grepTags='@login'`. Testovi koji sadrže *tag* bit će izvedeni, a testovi u istoj testnoj skripti koji ne sadrže *tag* bit će u statusu čekanja (engl. *pending*). Slika 4.19. prikazuje izvođenje skripte s testovima za prijavu korisnika.

```
Running:  login.cy.js                                                    (5 of 12)

Parabank Login API Test
  ✓ Should successfully login with correct credentials (438ms)
  - Should fail login with incorrect password
  - Should fail login with incorrect username
  - Should fail login with incorrect credentials (incorrect password and username)

1 passing (528ms)
3 pending

(Results)

┌──────────────┬──────────────┐
│ Tests:       │ 4            │
│ Passing:     │ 1            │
│ Failing:     │ 0            │
│ Pending:     │ 3            │
│ Skipped:     │ 0            │
│ Screenshots: │ 0            │
│ Video:       │ false        │
│ Duration:    │ 0 seconds    │
│ Spec Ran:    │ login.cy.js  │
└──────────────┴──────────────┘
```

Slika 4.19. *Rezultat izvođenja testa koji sadrži @login grep tag.*

5. ANALIZA REZULTATA TESTIRANJA

U ovom poglavlju analiziraju se rezultati manualnog i automatskog testiranja.

5.1. Rezultati automatskog testiranja

Na slici 5.1. prikazani su rezultati automatskog izvođenja testova. Iz priloženih rezultata se može vidjeti da API radi očekivano, odnosno u skladu s pripadajućom dokumentacijom.

	Spec		Tests	Passing	Failing	Pending	Skipped
✔	accountByID.cy.js	00:02	4	4	–	–	–
✔	accountsOverview.cy.js	00:01	4	4	–	–	–
✔	customerDetails.cy.js	00:01	3	3	–	–	–
✔	loanRequest.cy.js	00:01	3	3	–	–	–
✔	login.cy.js	00:01	4	4	–	–	–
✔	newCheckingAccount.cy.js	00:01	5	5	–	–	–
✔	newSavingAccount.cy.js	00:01	4	4	–	–	–
✔	payBill.cy.js	00:01	6	6	–	–	–
✔	transactionsByDate.cy.js	00:01	5	5	–	–	–
✔	transactionsForAccountByAmount.cy.js	00:01	5	5	–	–	–
✔	transferFunds.cy.js	00:02	9	9	–	–	–
✔	updateCustomerInfo.cy.js	805ms	2	2	–	–	–
✔	All specs passed!	00:18	54	54	–	–	–

Slika 5.1. *Rezultati pokretanja svih testova u terminalu.*

U slučaju da je neki od testova pao, potrebno je pronaći mjesto pada testa i analizirati razlog zbog kojeg je test pao. Moguća su dva razloga pada testova. Promjena u specifikacijama i zahtjevima na softver ili greška u radu softvera. Ako je razlog pada testa promjena zahtjeva, potrebno je popraviti testnu skriptu, to jest prilagoditi je trenutnim zahtjevima. S druge strane, ako je inženjer osiguranja kvalitete utvrdio da je riječ o grešci u radu, potrebno je napisati izvješće o grešci (engl. *bug report*). Cilj tog izvješća je pružiti informacije o tome kako je došlo do uočavanja greške. Izvješće o greškama sadrži podatke o serveru i verziji softvera na kojem je greška uočena, korake kojima se dolazi do dijela aplikacije u kojoj je greška uočena, kao i očekivani i stvarni rezultat tih koraka. To se izvješće šalje razvojnim inženjerima koji onda na temelju toga pronalaze uzrok te greške i popravljaju je. Nakon što razvojni inženjeri poprave grešku, mora se napraviti *deploy* kôda na server. Inženjer osiguranja kvalitete je dužan ponovno

testirati funkcionalnost u kojoj je pronađena greška kako bi se utvrdilo da funkcionalnost sada zaista ima očekivano ponašanje.

U drugom stupcu izvješća o rezultatima izvođenja testova prikazano je vrijeme izvođenja svake testne skripte kao i vrijeme izvođenja svih testova. U testnim slučajevima u Prilogu 2. navedena su 46 različita testna slučaja. Kako bi se osigurala neovisnost izvođenja testnih skripti neki od testnih slučajeva se ponavljaju pa je ukupan broj testova koji se izvode 54. U 12 testnih skripti nalazi se 54 testnih slučajeva, a za njihovo izvođenje je bilo potrebno 18 sekundi. Dakle, u prosjeku je izvođenje jedne testne skripte trajalo 1,5 sekundi.

5.2. Rezultati manualnog testiranja

Manualno izvođenje testnih slučajeva provedeno je koristeći Postman. Manualno je testirano 54 testnih slučajeva i njihovo izvođenje je trajalo 41 minutu. Prilikom manualnog izvođenja uočeno je da sesije ponekad traju samo nekoliko minuta. To dodatno otežava i usporava manualno izvođenje testova. Na kraju svake sesije korisnik je izbrisan iz baze podataka i potrebna je ponovna registracija.

5.3. Usporedba rezultata

U dolje priloženoj Tablici 5.1., nalazi se usporedba rezultata izvođenja testova.

Tablica 5.1. *Usporedba izvođenja automatskih i manualnih testova.*

	Automatsko izvođenje testova	Manualno izvođenje testova
Ukupan broj testova	54	54
Broj uspješno izvedenih testova	54	54
Broj neuspješno izvedenih testova	0	0
Vrijeme izvođenja	18 sekundi	41 minuta

Iz priloženih rezultata moguće je zaključiti da je izvođenje automatskih testova značajno brže od manualnog izvođenja. Rezultati testiranja su jednaki, odnosno broj pronađenih pogrešaka prilikom automatskog izvođenja testova i manualnog izvođenja je jednak. Dakle, automatizirani

testovi su jednako pouzdani kao i manualno izvođenje testova, a nekoliko desetaka puta brži. Jedinu manu koju automatizirano testiranje nije uspjelo uhvatiti je trajanje sesije koja bi se mogla smatrati pogreškom u radu sofvera. Razvojni inženjeri bi trebali istražiti razlog kratkog trajanja sesija i popraviti ga kako bi se smanjila mogućnost nezadovoljstva korisnika.

Iz svega je moguće zaključiti da iako sama automatizacija testova dugo traje, njezino izvođenje je značajno brže. Automatizacija testova može uštedjeti vrijeme kada se više puta izvodi testiranje istih funkcionalnosti. Tako bi se osiguralo da glavne funkcionalnosti softvera rade ispravno iako su implementirane promjene nekih drugih, povezanih funkcionalnosti.

6. ZAKLJUČAK

U ovom diplomskom radu obrađena je tema testiranja API-ja u Cypress-u. Testiranje je jedna od faza razvoja softvera kojoj je cilj osigurati da razvijeni softver radi prema zadanih zahtjevima. U ovom slučaju testiran je API. API omogućuje prijenos podataka u različitim formatima, a za prijenos koristi protokole za razmjenu podatka. S obzirom na to što se testira potrebno je odabrati prikladnu vrstu testiranja i alate koji će se koristiti.

Testni API je dio ParaBank testne aplikacije koja oponaša funkcionalnosti internet bankarstva. Za automatizirano testiranje API-ja korišteno je Cypress okruženje. Cypress je jedan od često korištenih alata za automatizaciju testiranja, a testne skripte pišu se u JavaScript programskom jeziku. U radu je objašnjena važnost testiranja softvera i neke od vrsta testiranja API-ja. Predstavljeni su osnovni principi rada API-ja, slanje zahtjeva/odgovora, statusni kodovi i formati u kojima API prenosi podatke od klijenta prema serveru i obrnuto. Također, navedeni su i opisani neki od često korištenih alata za automatizaciju testova. Zatim je opisano cijelo razvojno okruženje u kojima su pisane skripte za automatizirano izvođenje testova. Na kraju su analizirani rezultati automatskog i manualnog testiranja i uspoređeni su njihovi rezultati.

Analizom rezultata zaključeno je da je automatsko testiranje jednako pouzdano kao i manualno testiranje softvera, a prednost automatskog testiranja je to što se može izvesti u vrlo kratkom periodu što nije slučaj s manualnim testiranjem. Međutim, automatizacija testova vrlo je skup i dugotrajan proces i potrebno je osoblje koje zna pisati programski kôd. Zbog toga se automatizacija testova provodi u slučajevima kada se testiranja istih komponenti ponavljaju jer se tada isplati ulaganje u automatizaciju. Ni automatski testovi ne mogu opstati bez intervencija inženjera osiguranja kvalitete. Zbog promjena zahtjeva na softver, skripte za automatske testove je potrebno održavati i mijenjati ih kako bi testiranje i dalje davalo valjane rezultate. Manualno testiranje pogodno je za testiranje komponenti za čiju procjenu je potreban čovjek, kao kada se testira uporabljivost softvera ili izgled korisničkog sučelja. Iz svega se može zaključiti da je kombinacija manualnog i automatskog testiranja ključ razvoja uspješnog softvera.

Iako se iz rezultata testiranja može zaključiti da API radi u skladu sa pripadajućom dokumentacijom, automatizaciju testiranja moguće je proširiti. Testiranje administratorskih *endpoint-ova* nije obuhvaćeno ovim diplomskim radom. Njihovim testiranjem povećala bi se pokrivenost testiranja, a tako bi se mogla utvrditi i ispravnost cjelokupnog API-ja.

LITERATURA

[1] A. Mishra and D. Dubey, "A comparative study of different software development life cycle models in different scenarios", International Journal of Advance research in computer science and management studies, str. 65, 2013., dostupno na: https://www.researchgate.net/publication/289526047_A_Comparative_Study_of_Different_Software_Development_Life_Cycle_Models_in_Different_Scenarios [9.9.2024.]

[2] E. Y. Li, "Software testing in a system development process: A life cycle perspective", Journal of Systems Management 41.8 (1990): 23-31., str. 3, 1990., dostupno na: https://www.researchgate.net/publication/255650176_Software_Testing_In_A_System_Development_Process_A_Life_Cycle_Perspective

[3] L. G. Hayes, "The automated testing handbook", Software Testing Institute, str. 9, 2004., dostupno na: https://books.google.de/books?id=-jangThcGIkC&dq=automated+testing&lr=&hl=hr&source=gbs_navlinks_s [9.9.2024.]

[4] J. Alahari, et al., "Implementing Continuous Integration/Continuous Deployment (CI/CD) Pipelines for Large-Scale iOS Applications", Darpan International Research Analysis 12.3 (2024), 522-534, str. 525, 2024., dostupno na: https://dira.shodhsagar.com/index.php/j/article/view/104 [9.9.2024.]

[5] N. Islam, "A Comparative Study of Automated Software Testing Tools", Culminating Projects in Computer Science and Information Technology, str. 8, 2016., dostupno na: https://repository.stcloudstate.edu/csit_etds/12/ [1.9.2024.]

[6] K. Bahl, "Software Testing Tools & Techniques for Web Applications" no. 5, pp. 315– 318, 2015. 2.1

[7] R. L. Black, E. Van Veenendaal, D. Graham, „Foundations of Software Testing: ISTQB certification", 4th Edition, str. 46-48, 2021.

[8] Matthias Biehl, "API Architecture; The Big Pictures For Building APIs", Vol. 2. API-University Press, str. 16-18, 2015. dostupno na: https://books.google.hr/books?id=6D64DwAAQBAJ [1.9.2024.]

[9] D. Peng† , L. Cao, W. Xu, “Using JSON for Data Exchanging in Web Service Applications”, Journal of Computational Information Systems 7.16 (2011): 5883-5890., str. 15886, 15889, 2011., dostupno na: https://www.researchgate.net/publication/265874991_Using_JSON_for_Data_Exchanging_in_Web_Service_Applications [1.9.2024.]

[10] D. Gourley, B. Totty, M. Sayer, A. Aggarwal, S. Reddy, “HTTP: The Definitive Guide”, O'Reilly Media, Inc., str. 48-50, 53-57, 2002., dostupno na: https://www.google.de/books/edition/HTTP_The_Definitive_Guide/qEoOl9bcV_cC?hl=hr&gbpv=0 [1.9.2024.]

[11] L. Richardson, S. Ruby, “RESTful Web Services”, O'Reilly Media, Inc., str. 371-373, 2008., dostupno na: https://www.google.de/books/edition/RESTful_Web_Services/XUaErakHsoAC?hl=hr&gbpv=0 [1.9.2024.]

[12] R. Angmo and M. Sharma, "Performance evaluation of web based automation testing tools," *2014 5th International Conference - Confluence The Next Generation Information Technology Summit (Confluence)*, Noida, India, pp. 731-735, doi: 10.1109/CONFLUENCE.2014.6949287., 2014, dostupno na: https://ieeexplore.ieee.org/abstract/document/6949287 [1.9.2024.]

[13] Why Cypress? dostupno na: https://docs.cypress.io/guides/overview/why-cypress [1.9.2024.]

[14] How Does Appium Work?, dostupno na: https://appium.io/docs/en/2.6/intro/appium/ [1.9.2024.]

[15] The Selenium Browser Automation Project, dostupno na: https://www.selenium.dev/documentation/ [1.9.2024.]

[16] Postman docmentation overview, dostupno na: https://learning.postman.com/docs/introduction/overview/ [1.9.2024.]

[17] L. Baresi, M. Pezz`e, “An Introduction to Software Testing”, *Electronic Notes in Theoretical Computer Science* 148.: 89-111., str. 92, 2006, dostupno na: https://www.sciencedirect.com/science/article/pii/S1571066106000442?ref=pdf_download&fr=RR-2&rr=89f91d6b7d5b90e2 [1.9.2024.]

[18] Visual Studio Code Overview, dostupno na: https://code.visualstudio.com/docs [1.9.2024.]

[19] About npm, dostupno na: https://docs.npmjs.com/about-npm [1.9.2024.]

[20] Introduction to Node.js, dostupno na: https://nodejs.org/en/learn/getting-started/introduction-to-nodejs [1.9.2024.]

[21] Chai assertion library, dostupno na: https://www.chaijs.com/ [1.9.2024.]

[22] Hooks, dostupno na: https://mochajs.org/#hooks [1.9.2024.]

[23] Writing and Organizing Tests, dostupno na: https://docs.cypress.io/guides/core-concepts/writing-and-organizing-tests [1.9.2024.]

[24] Fixture, dostupno na: https://docs.cypress.io/api/commands/fixture [1.9.2024.]

Sažetak

Cilj ovog diplomskog rada je automatizacija testiranja API-ja koji je dio testnog softvera Parabank. Objašnjena je važnost testiranja softvera i princip rada API-ja. Detaljno je opisan postupak provođenja testiranja koji započinje planiranjem testiranja i odabirom alata za testiranje. Objašnjeno je i kako se pišu skripte za automatsko izvođenje testova u Cypress okruženju. Alat korišten za automatizaciju testova je Cypress, a testne skripte pisane su programskim jezikom JavaScript. Manualno izvođenje testnih slučajeva provedeno je koristeći Postman. Napravljena je analiza izvođenja automatskih i manualnih testova, što uključuje vrijeme izvođenja kao i rezultate izvedenih testova. Objašnjene su prednosti i mane obje vrste testiranja i utjecaj njihove kombinacije na kvalitetu softvera. Na kraju je napravljen zaključak o isplativosti ulaganja u automatsko testiranje.

Ključne riječi: API, automatizacija, Cypress, osiguranje kvalitete, testiranje

Abstract

Title: Automated API testing using the Cypress framework

The goal of this thesis is automatization of APIs testing which is the part of testing software called ParaBank. Importance of software testing and API principles are explained. Details of the testing process, which starts with testing planning and choice of testing tools, are described. Also, the process of writing test scripts using the Cypress framework is explained. Cypress is a tool used for test automatization and test scripts are written in the programming language JavaScript. Manual testing is executed using Postman. Analysis of automated and manual testing is made, including execution time and results of test execution. Explanation of advantages and disadvantages of both testing types are given, as well as influence of their combination on software quality. In the end, the conclusion about cost-effectiveness of test automation is made.

Key words: API, automatization, Cypress, quality assurance, testing

Životopis

Dejana Ristić, rođena 16.12.1999. u Osijeku. U Belom Manastiru je završila osnovnu školu “Dr. Franjo Tuđman”, a školske godine 2013./2014. upisuje srednju školu Gimnazija Beli Manastir. Nakon završetka srednjoškolskog obrazovanja upisuje preddiplomski sveučilišni studij Računarstvo na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija. 2022. godine završava preddiplomski studij i upisuje sveučilišni diplomski studij Računarstva, izborni blok Programsko inženjerstvo.

Prilozi

Prilog 1. Programski kôd

https://github.com/dejana-ferit99/diplomski_1.git

Prilog 2. Testni slučajevi

ParaBank API dokumentacija: https://parabank.parasoft.com/parabank/api-docs/index.html

Naziv datoteke	Oznaka testnog slučaja	HTTP metoda	URL	Opis testnog slučaja	Testni
login.cy.js	TC-1	POST	http://parabank.parasoft.com/parabank/services/bank/login/:username/:password	Prijava korisnika s ispravnim *username* i *password* podacima	userna passw
login.cy.js	TC-2	POST	http://parabank.parasoft.com/parabank/services/bank/login/:user	Prijava korisnika s ispravnim *username*	userna invalic

			name/:password	podatkom i neispravnim *password* podatkom	
login.cy.js	TC-3	POST	http://parabank.parasoft.com/parabank/services/bank/login/:username/:password	Prijava korisnika s neispravnim *username* podatkom i ispravnim *password* podatkom	invalid passw
login.cy.js	TC-4	POST	http://parabank.parasoft.com/parabank/services/bank/login/:username/:password	Prijava korisnika s neispravnim *username* i *password* podatkom	invalid invalid
accountsOverview.cy.js	TC-5	GET	https://parabank.parasoft.com/parabank/services_proxy/bank/customers/${customerID}/accounts	Dohvaćanje svih bankovnih računa korisnika koristeći ispravan *customerID* podatak i autorizaciju	custom
accountsOverview.cy.js	TC-6	GET	https://parabank.parasoft.com/parabank/services_proxy/bank/customers/${customerID}/account	Dohvaćanje svih bankovnih računa korisnika koristeći	custom

			s	ispravan *customerID* podatak, bez autorizacije	
accountsOverview.cy.js	TC-7	GET	https://parabank.parasoft.com/parabank/services_proxy/bank/customers/${invalidID}/accounts	Dohvaćanje svih bankovnih računa korisnika koristeći neispravan *customerID* podatak	invalid
accountsOverview.cy.js	TC-8	GET	https://parabank.parasoft.com/parabank/services_proxy/bank/customers/accounts	Dohvaćanje svih bankovnih računa korisnika bez korištenja *customerID* podatka	
accountByID.cy.js	TC-9	GET	https://parabank.parasoft.com/parabank/services_proxy/bank/accounts/${accountID}	Dohvaćanje detalja bankovnog računa koristeći ispravan *accountID* podatak	accoun
accountByID.cy.js	TC-10	GET	https://parabank.parasoft.com/parabank/services_proxy/bank/accounts/${invalidID}	Dohvaćanje detalja bankovnog računa bez korištenja *accountID* podatka	

accountByID.cy.js	TC-11	GET	https://parabank.parasoft.com/parabank/services_proxy/bank/accounts/	Dohvaćanje detalja bankovnog računa koristeći neispravan *accountID*	invalid
transactionsForAccountByAmount.cy	TC-12	GET	https://parabank.parasoft.com/parabank/services_proxy/bank/accounts/${accountID}/transactions/amount/500	Kreiranje transakcije koristeći *accountID* i *amount* podatak	accoun
transactionsForAccountByAmount.cy	TC-13	GET	https://parabank.parasoft.com/parabank/services_proxy/bank/accounts/${accountID}/transactions/amount/	Kreiranje transakcije koristeći *accountID* i bez korištenja *amount* podataka	accoun
transactionsForAccountByAmount.cy	TC-14	GET	https://parabank.parasoft.com/parabank/services_proxy/bank/accounts/${invalidID}/transactions/amount/	Kreiranje transakcije koristeći neispravan *accountID* i bez korištenja *amount* podataka	invalid
transactionsForAccountByAmount.cy	TC-15	GET	https://parabank.parasoft.com/parabank/services_proxy/bank/accounts/${invalidID}/transactions/amount/500	Kreiranje transakcije koristeći neispravan *accountID* i koristeći *amount* podatak	invalid amoun
transferFunds.cy.js	TC-16	POST	https://parabank.parasoft.com/parabank/services_proxy/bank/transfer?fromAccountId=${accountID}&toAccountId=${accountID}&amount=500	Prijenos novca s jednog računa na drugi koristeći ispravan *fromAccountID*,	fromA toAcc amoun

				ispravan *toAccountID* i *amount* podatak	
transferFunds.cy.js	TC-17	POST	https://parabank.parasoft.com/parabank/services_proxy/bank/transfer?fromAccountId=${accountID}&toAccountId=${accountID}&amount=	Prijenos novca s jednog računa na drugi koristeći ispravan *fromAccountID*, ispravan *toAccountID* i bez *amount* podataka	fromA toAcc
transferFunds.cy.js	TC-18	POST	https://parabank.parasoft.com/parabank/services_proxy/bank/transfer?fromAccountId=${invalidID}&toAccountId=${accountID}&amount=500	Prijenos novca s jednog računa na drugi koristeći neispravan *fromAccountID*, ispravan *toAccountID* i *amount* podatak	invalid fromA toAcc amoun
transferFunds.cy.js	TC-19	POST	https://parabank.parasoft.com/parabank/services_proxy/bank/transfer?fromAccountId=${accountID}&toAccountId=${invalidID}&amount=500	Prijenos novca s jednog računa na drugi koristeći ispravan *fromAccountID*, neispravan *toAccountID* i *amount* podatak	fromA invalid toAcc amoun

transferFunds.cy.js	TC-20	POST	https://parabank.parasoft.com/parabank/services_proxy/bank/transfer?fromAccountID=${accountID}&toAccountId=${invalidID}&amount=	Prijenos novca s jednog računa na drugi koristeći ispravan *fromAccountID*, neispravan *toAccountID* i bez *amount* podatka	fromA invalid toAcc
transferFunds.cy.js	TC-21	POST	https://parabank.parasoft.com/parabank/services_proxy/bank/transfer?fromAccountId=${invalidID}&toAccountId=${invalidID}&amount=500	Prijenos novca s jednog računa na drugi koristeći neispravan *fromAccountID*, neispravan *toAccountID* i *amount* podatak	invalid fromA invalid toAcc amoun
transferFunds.cy.js	TC-22	POST	https://parabank.parasoft.com/parabank/services_proxy/bank/transfer?fromAccountId=${invalidID}&toAccountId=${invalidID}&amount=	Prijenos novca s jednog računa na drugi koristeći neispravan *fromAccountID*, neispravan *toAccountID* i bez *amount* podatka	invalid fromA invalid toAcc

transferFunds.cy.js	TC-23	POST	https://parabank.parasoft.com/parabank/services_proxy/bank/transfer?fromAccountId=${accountID}&toAccountId=${invalidID}&amount=	Prijenos novca s jednog računa na drugi koristeći neispravan *fromAccountID*, ispravan *toAccountID* i bez *amount* podatka	invalid fromA toAcc
payBill.cy.js	TC-24	POST	https://parabank.parasoft.com/parabank/services_proxy/bank/billpay?accountId=${accountID}&amount=200	Plaćanje računa koristeći ispravan *accountID*, *amount* podatak i *body*	valid a amoun "name "addre "street "city": "state" "zipCo "string }, "phone "string "accou 0 }
payBill.cy.js	TC-25	POST	https://parabank.parasoft.com/parabank/services_proxy/bank/billpay?accountId=${accountID}&amount=200	Plaćanje računa koristeći ispravan *accountID*, *amount* podatak i ne koristeći *body*	accoun

payBill.cy.js	TC-26	POST	https://parabank.parasoft.com/parabank/services_proxy/bank/billpay?accountId=${accountID}&amount=	Plaćanje računa koristeći ispravan *accountID* i *body*, bez korištenja *amount* podatka	valid a body: "name "addre "street "city": "state" "zipCo "string }, "phone "string "accou 0 }
payBill.cy.js	TC-27	POST	https://parabank.parasoft.com/parabank/services_proxy/bank/billpay?accountId=${invalidID}&amount=	Plaćanje računa koristeći neispravan *accountID*, bez korištenja *amount* podataka i koristeći *body*	invalid body: "name "addre "street "city": "state" "zipCo "string }, "phone "string

					"accou 0 }
payBill.cy.js	TC-28	POST	https://parabank.parasoft.com/parabank/services_proxy/bank/billpay?accountId=${accountID}&amount=200	Plaćanje računa koristeći ispravan *accountID* podatak, *amount* podatak i *body* kojem nedostaje podatak *accountNumber*	body: "name "addre "street "city": "state" "zipCo "string }, "phone "string "accou 0 }

transactionsByDate.cy.js	TC-29	GET	https://parabank.parasoft.com/parabank/services_proxy/bank/accounts/${accountID}/transactions/onDate/${todayDateString}	Dohvati transakcije računa koristeći ispravan *accountID* i *date* podatak	accour
transactionsByDate.cy.js	TC-30	GET	https://parabank.parasoft.com/parabank/services_proxy/bank/accounts/${accountID}/transactions/onDate/3-23-2024	Dohvati transakcije računa koristeći ispravan *accountID* i koristeći *date* podatak za koji ne postoje transakcije	accour date
transactionsByDate.cy.js	TC-31	GET	https://parabank.parasoft.com/parabank/services_proxy/bank/accounts/${invalidID}/transactions/onDate/${todayDateString}	Dohvati transakcije računa koristeći neispavan *accountID* i *date* podatak	invalid date

transactionsByDate.cy.js	TC-32	GET	https://parabank.parasoft.com/parabank/services_proxy/bank/accounts/${accountID}/transactions/onDate/	Dohvati transakcije računa koristeći ispravan *accountID* i ne koristeći *date* podatak	accour
newCheckingAccount.cy.js	TC-33	POST	https://parabank.parasoft.com/parabank/services_proxy/bank/createAccount?customerId=${customerID}&newAccountType=0&fromAccountId=${accountID}	Kreiraj novi tekući račun (*checking account*) koristeći ispravan *customerID*, *AccountType* i *fromAccountID* podatak	custon Accou fromA
newCheckingAccount.cy.js	TC-34	POST	https://parabank.parasoft.com/parabank/services_proxy/bank/createAccount?customerId=${customerID}&newAccountType=-1&fromAccountId=${invalidID}	Kreiraj novi *checking account* koristeći ispravan *customerID*, neispravan *AccountType* i neispravan *fromAccountID* podatak	custon invalic Accou invalic fromA
newCheckingAccount.cy.js	TC-35	POST	https://parabank.parasoft.com/parabank/services_proxy/bank/createAccount?customerId=${invalidID}&newAccountType=-1	Kreiraj novi *checking account* koristeći neispravan *customerID*,	invalic custon invalic Accou

			&fromAccountId=${accountID}	neispravan *AccountType* i ispravan *fromAccountID* podatak	fromA
newCheckingAccount.cy.js	TC-36	POST	https://parabank.parasoft.com/parabank/services_proxy/bank/createAccount?customerId=${invalidID}&newAccountType=0&fromAccountId=${invalidID}	Kreiraj novi *checking account* koristeći neispravan *customerID*, ispravan *AccountType* i neispravan *fromAccountID* podatak	invalid custom Accou invalid fromA
newSavingAccount.cy.js	TC-37	POST	https://parabank.parasoft.com/parabank/services_proxy/bank/createAccount?customerId=${customerID}&newAccountType=1&fromAccountId=${accountID}	Kreiraj novi *saving account* (štedni račun) koristeći ispravan *customerID*, *AccountType* i *fromAccountID* podatak	custom Accou fromA
newSavingAccount.cy.js	TC-38	POST	https://parabank.parasoft.com/parabank/services_proxy/bank/createAccount?customerId=${customerID}&newAccountType=1	Kreiraj novi *saving account* koristeći ispravan *customerID*, ispravan *AccountType*	custom Accou invalid fromA

			&fromAccountId=${invalidID}	i neispravan *fromAccountID* podatak	
newSavingAccount.cy.js	TC-39	POST	https://parabank.parasoft.com/parabank/services_proxy/bank/createAccount?customerId=${customerID}&newAccountType=-1&fromAccountId=${invalidID}	Kreiraj novi *saving account* koristeći ispravan *customerID*, neispravan *AccountType* i neispravan *fromAccountID* podatak	custom invalid Accou invalid fromA
customerDetails.cy.js	TC-40	GET	http://parabank.parasoft.com/parabank/services_proxy/bank/customers/${customerID}	Dohvati podatke korisnika koristeći ispravan *customerID* podatak	custom

customerDetails.cy.js	TC-41	GET	http://parabank.parasoft.com/parabank/services_proxy/bank/customers/	Dohvati podatke korisnika bez korištenja *customerID* podatka	
customerDetails.cy.js	TC-42	GET	http://parabank.parasoft.com/parabank/services_proxy/bank/customers/${invalidID}	Dohvati podatke korisnika koristeći neispravan *customerID* podatak	invalid
updateCustomerInfo.cy.js	TC-43	POST	https://parabank.parasoft.com/parabank/services_proxy/bank/customers/update/${customerID}?firstName=User&lastName=Useric&street=Usely&city=Osijek&state=Osjecko-baranjska&zipCode=31000&phoneNumber=0987654321&ssn=1234567890&username=${username}&password=${password}	Izmjeni podatke korisnika koristeći ispravan *customerID* podatak i *firstName*, *lastName*, *street*, *city*, *state*, *zipCode*, *phoneNumber*, *ssn*, *username*, *password* podatke	custom firstNa lastNa city, st zipCo phone userna passw

updateCustomerInfo.cy.js	TC-44	POST	https://parabank.parasoft.com/parabank/services_proxy/bank/customers/update/${invalidID}?firstName=User&lastName=User ic&street=Usely&city=Osijek&state=Osjecko-baranjska&zipCode=31000&phoneNumber=0987654321&ssn=1234567890&username=${username}&password=${password}	Izmjeni podatke korisnika koristeći ispravan *customerID* podatak i *firstName*, *lastName*, *street*, *city*, *zipCode*, *phoneNumber*, *ssn*, *username*, *password podatke*, *state* podatak se ne koristi	custom firstNa lastNa city, zi phonel userna passwo
loanRequest.cy.js	TC-45	POST	https://parabank.parasoft.com/parabank/services_proxy/bank/requestLoan?customerId=${customerID}&amount=100&downPayment=10&fromAccountId=${accountID}	Pošalji zahtjev za zajam s dovoljnim novčanim sredstvima	custom amoun downP fromA

loanRequest.cy.js	TC-46	POST	https://parabank.parasoft.com/parabank/services_proxy/bank/requestLoan?customerId=${customerID}&amount=10000&downPayment=100&fromAccountId=${accountID}	Pošalji zahtjev za zajam s nedovoljnim iznosom otplate	custom amoun downP fromA