# Automatizirano smoke testiranje internet aplikacija

**Vorgić, Amela**



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

**Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit**

**Osijek, 15.09.2023.**

**Odboru za završne i diplomske ispite**

# Imenovanje Povjerenstva za diplomski ispit

| | |
|---|---|
| **Ime i prezime Pristupnika:** | Amela Vorgić |
| **Studij, smjer:** | Diplomski sveučilišni studij Računarstvo |
| **Mat. br. Pristupnika, godina upisa:** | D-1256R, 07.10.2021. |
| **OIB studenta:** | 43138304295 |
| **Mentor:** | doc. dr. sc. Ivan Vidović |
| **Sumentor:** | , |
| **Sumentor iz tvrtke:** | Nikolina Mihić |
| **Predsjednik Povjerenstva:** | prof. dr. sc. Goran Martinović |
| **Član Povjerenstva 1:** | doc. dr. sc. Ivan Vidović |
| **Član Povjerenstva 2:** | izv. prof. dr. sc. Tomislav Matić |
| **Naslov diplomskog rada:** | Automatizirano smoke testiranje internet aplikacija |
| **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 internet aplikacija. Nakon provedenog istraživanja potrebno je opisati Cypress okruženje te ga usporediti s ostalim postojećim rješenjima. U praktičnom dijelu rada potrebno je primjeniti Cypress okruženje za provedbu automatiziranih smoke testova na odabranoj internet aplikaciji. Tema rezervirana za: Amela Vorgić Sumentor iz tvrtke: Nikolina Mihić (COBE) |
| **Prijedlog ocjene pismenog dijela ispita (diplomskog rada):** | Izvrstan (5) |
| **Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:** | Primjena znanja stečenih na fakultetu: 3 bod/boda<br>Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda<br>Jasnoća pismenog izražavanja: 3 bod/boda<br>Razina samostalnosti: 3 razina |
| **Datum prijedloga ocjene od strane mentora:** | 15.09.2023. |

| | |
|---|---|
| Potvrda mentora o predaji konačne verzije rada: | *Mentor elektronički potpisao predaju konačne verzije.* |
| | Datum: |

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

# IZJAVA O ORIGINALNOSTI RADA

**Osijek, 02.10.2023.**

| | |
|---|---|
| **Ime i prezime studenta:** | Amela Vorgić |
| **Studij**: | Diplomski sveučilišni studij Računarstvo |
| **Mat. br. studenta, godina upisa:** | D-1256R, 07.10.2021. |
| **Turnitin podudaranje [%]:** | 2 |

Ovom izjavom izjavljujem da je rad pod nazivom**: Automatizirano smoke testiranje internet aplikacija**

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 studenta:

**SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU**
**FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK**

**Sveučilišni studij**

# Automatizirano smoke testiranje internet aplikacija

**Diplomski rad**

**Amela Vorgić**

**Osijek, 2023.**

# SADRŽAJ

# 1. UVOD

Digitalizacija života dovela je do toga da se većina svakodnevnih zadaća može obaviti putem interneta. Za komunikaciju korisnika s internetom potrebno je koristiti različite internet aplikacije koje pružaju grafičko sučelje kako bi prosječan korisnik mogao upotrebljavati funkcionalnosti koje aplikacija pruža bez poznavanja unutarnje strukture aplikacije. Internet aplikacije više nisu ograničene samo za stolna i prijenosna računala jer postoji pregršt elektroničkih uređaja koji omogućuju pristup internetu i korištenje internet aplikacija. Razvoj internet aplikacija zahtjeva pomno planiranje čitavog procesa kako bi se ljudski i financijski resursi optimalno raspodijelili. Razvojni tim sastoji se od razvojnih programera, projekt menadžera i inženjera osiguranja kvalitete (engl. *Quality Assurance Engineer*). Inženjeri osiguranja kvalitete provode testiranja aplikacije kako bi se greške u radu pronašle i otklonile prije nego korisnici dođu u doticaj s aplikacijom. Osim manualnog testiranja, inženjeri osiguranja kvalitete pišu i programski kôd za automatizaciju testiranja kako bi se testiranje moglo izvoditi brže, temeljitije i kvalitetnije. Računalo se za razliku od čovjeka ne može umoriti pa može provesti iscrpnije testiranje bez zamora [10]. Automatiziranim testovima može se provoditi testiranje po uzoru na korisničko ponašanje i korištenje aplikacije, ali i testiranje *backend-a* i *frontend-a* bez testiranja funkcionalnosti, isključivo testirajući internet aplikaciju prema dizajnu. Automatizacija testiranja sve je popularnija pa postoji i sve više alata za provedbu testova, a alat se odabire prema funkcionalnostima aplikacije i onome što treba testirati.

U prvom poglavlju napravljen je uvod u tematiku testiranja kao i razlozi zbog kojih se uvodi automatizacija testiranja. Nakon uvodnog poglavlja, u drugom poglavlju predstavljen je pregled područja gdje je opisana nužnost testiranja internet aplikacija kao i strategija i plan pristupa testiranju. Testiranje je opisano prema vrstama koje opisuju krajnji cilj testiranja. Opisana je i struktura internet aplikacija i trenutno popularni alati za provedbu automatizacije testiranja internet aplikacija. Treće poglavlje opisuje pristup automatiziranom testiranju internet aplikacija te su objašnjeni odabrani programski jezik JavaScript, alat za automatizaciju Cypress i biblioteka Faker kao i priprema okruženja za pisanje automatiziranih testova. U četvrtom poglavlju je prikazana implementacija testnih slučajeva za internet aplikaciju ParaBank u obliku programskog kôda uz dodatno pojašnjenje *spec* datoteka koje sadrže testove i modela objekata stranica s lokatorima i metodama za komuniciranje s elementima. Prikazano je provođenje automatiziranih testova te analiza rezultata i usporedba manualnog i automatiziranog testiranja.

U petom poglavlju napravljena je analiza i usporedba rezultata manualnog i automatiziranog testiranja.

# 2. PREGLED PODRUČJA

Tehnološki napredak neprestano doprinosi razvoju i poboljšanju alata za pisanje automatiziranih testova. Automatizirani testovi omogućavaju kontinuirano i sistematsko testiranje ponašanja i rada aplikacije bez potrebe za stalnim ljudskim nadzorom i resursima [1]. Provođenje automatiziranih testova u početku zahtjeva veća ulaganja, ali s pravilnim održavanjem dugoročno su isplativiji. Složeni scenariji, kao i oni koji zahtijevaju različite ponavljajuće radnje, s velikom se preciznošću i brzinom mogu testirati korištenjem automatiziranih testova [2].

Postoji pregršt opcija za pristup internet aplikacijama s obzirom na vrstu uređaja, operativnog sustava i preglednika. Automatiziranim testovima isti testovi mogu se primijeniti na velik raspon različitih kombinacija kojima će se rad aplikacije ispitati u uvjetima nalik stvarnima [1]. Budući da se velik broj postojećih aplikacija neprestano nadograđuje, važno je konstantno pratiti rad postojećih funkcionalnosti. Integriranje testova u sustav kontinuirane integracije i isporuke (CI/CD) (engl. *Continuous Integration and Continuous Delivery)* postalo je standardna praksa u razvoju softvera jer pruža mogućnost da se sa svakom izmjenom kôda testovi automatizirano izvedu te se uz detaljan izvještaj potvrdi ispravan rad aplikacije ili eventualno odstupanje.

U ovom se poglavlju objašnjava potreba za testiranjem kao i vrste testiranja koja se mogu provoditi uz iscrpnu analizu alata za provođenje automatiziranih testova za internet aplikacije.

## 2.1. Nužnost testiranja

Cilj izrade aplikacije jest da do krajnjeg korisnika dođe proizvod u svom najboljem obliku, kako bi korisnik bez poteškoća mogao koristiti sve funkcionalnosti koje aplikacija nudi, a da je pri tome aplikacija stabilna, sigurna i kvalitetna. Prema [3], jedan od ciljeva testiranja je evaluirati stavke poput zahtjeva na softver, *user stories,* dizajna i programskog kôda kako bi se potvrdilo da su tražene karakteristike zadovoljene. Ukoliko je inženjer osiguranja kvalitete (engl. *Quality Assurance Engineer*) uključen u proces razvoja softvera od samog početka, uključujući već i pregled zahtjeva na softver, mogu se izbjeći potencijalne greške u kasnijim fazama [4].

Testiranje aplikacije tijekom razvoja osigurava visoku kvalitetu krajnjeg proizvoda jer se korisnik neće suočiti s raznim problemima koji se otkrivaju prilikom testiranja i otklanjaju prije nego što aplikacija dođe u dodir s korisnikom. Osim ukupnog dojma o pouzdanosti i ispravnosti aplikacije, testiranje prilikom razvoja osigurava i niže troškove održavanja aplikacije jer će doći do manje nepredviđenih pogrešaka u radu aplikacije na produkcijskoj platformi što ostavlja više

resursa za nove značajke i poboljšanje performansi. Osim što se testiranjem pronalaze eventualne pogreške u radu sustava, često se pruža i povratna informacija o mogućnostima poboljšanja aplikacije što dovodi do neprestanih ulaganja u unapređenje softvera [4].

## 2.2. Strategija testiranja

Prije početka testiranja važno je postaviti strategiju testiranja. Strategija testiranja značajan je čimbenik u postavljanju granica kojih se treba držati prilikom testiranja te nudi temelje za postavljanje testnog plana. Prilikom postavljanja strategije testiranja važno je imati informacije o aplikaciji za koju se testiranje provodi kako bi se na adekvatan način pristupilo testiranju. Ne postoji jedinstvena i ,,najbolja" strategija testiranja. Za svaki se proizvod treba razviti specifična strategija koja najbolje odgovara tom proizvodu. Strategija treba biti razvijena tako da sadrži raznolike tehnike i pristupe kako bi se greške koje nisu uhvaćene na jedan način uspjele pronaći na drugi, ali isto tako strategija treba biti izvediva i praktična [3].

## 2.3. Testni plan

Planiranje testiranja obuhvaća definiranje ciljeva testiranja kao i način pristupa testiranju. Prilikom izrade testnog plana važno je utvrditi kontekst i ograničenja kako bi se aplikacija što efikasnije testirala. Prema [5], testni plan može se shvatiti kao ,,recept" po kojem će se provoditi testiranje. Testnim planom definiraju se opseg, kriteriji i potencijalni rizici kako bi se prije samog procesa testiranja utvrdilo na što treba obratiti pozornost.

## 2.4. Vrste testiranja

Prema [3], vrste testiranja uvode se kao sredstvo za jasno definiranje cilja određene razine testiranja za program. Vrsta testiranja usmjerena je na određeni cilj testa te se prema cilju odlučuje na koji način će se testiranje izvoditi.

### 2.4.1. Funkcionalno testiranje

Funkcionalno testiranje odnosi se na testiranje funkcionalnosti softvera, odnosno na to što bi softver trebao moći raditi. Funkcionalni zahtjevi na softver trebaju biti određeni već u *user stories* te se funkcionalni testovi i testni slučajevi kojima se pokriva ova vrsta testiranja temelje na zahtijevanim funkcionalnostima. Temeljitost funkcionalnog testiranja može se iskazati u postotku pokrivenosti elemenata čija se funkcionalnost može prikazati [3]. Na primjer, ukoliko u navigacijskoj traci postoji 5 elemenata za odabir i prilikom testiranja se izvede po jedan test za

svaki element, može se reći da se testovima pokrilo 100% opcija. Ovakvo iskazivanje ne znači da je komponenta 100% testirana, ali pokazuje da se svaki element komponente testirao barem u jednom kontekstu.

### 2.4.2. Nefunkcionalno testiranje

Vrsta testiranja koja pokriva nefunkcionalne atribute softvera naziva se nefunkcionalno testiranje. Nefunkcionalnim testiranjem utvrđuje se kako softver radi izvan okvira zahtijevanih funkcionalnosti i provodi se na svim razinama testiranja [3]. Neke od karakteristika koje se testiraju su brzina izvođenja, sigurnost, pouzdanost i održivost, a svaka od ovih karakteristika na svoj način pridonosi kvaliteti i uporabljivosti softvera. Internet aplikacije generalno trebaju zadovoljavati nefunkcionalne zahtjeve vezane uz uređaje i platforme putem kojih im se pristupa. Brzina odziva na zahtjev te velik broj korisnika u isto vrijeme neke su od ključnih značajki koje se testiraju izvan funkcionalnih zahtjeva. Uzme li se primjer internet aplikacije za bankarstvo, vrlo je važno da su korisnički podaci sigurni i zaštićeni od neovlaštenog pristupa, a isto tako važno je da se korisničkom računu može pristupiti neovisno o dobu dana i lokaciji jer korisnik može zatražiti usluge i informacije bilo kada i bilo gdje.

### 2.4.3. Strukturno testiranje

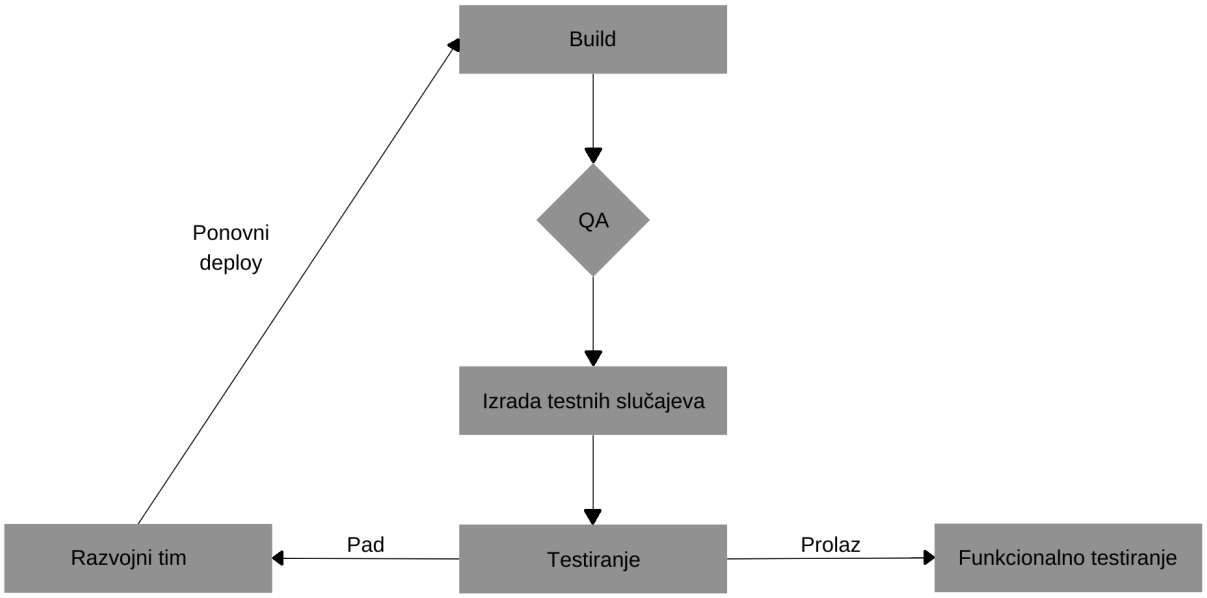
Strukturno testiranje naziva se još i *white-box* testiranje, a ovom vrstom testiranja zadire se u unutarnju strukturu sustava i implementaciju programskog kôda. Testira se arhitektura sustava, struktura programskog kôda kao i struktura podataka. Za strukturno testiranje važno je poznavati kôd i logiku implementacije kako bi se rezultati testiranja mogli analizirati. Osim inženjera osiguranja kvalitete, strukturno testiranje kôda moraju provoditi i razvojni programeri [3].

### 2.4.4. Regresijsko testiranje

Regresijskim testiranjem testiraju se već testirani slučajevi te za cilj ima potvrditi da nema regresije u rezultatima. Testiraju se sve važne funkcionalnosti, ali bez zadiranja u detalje. Prilikom svake izmjene kôda, potrebno je izvesti regresijsko testiranje kako bi se dokazalo da program i dalje radi očekivano. Osim testiranja komponenti koje su se mijenjale, važno je testirati i sve ostale kako bi se izbjeglo nenamjerno narušavanje funkcionalnosti koje nisu izravno povezane, ali u određenim okolnostima utječu jedna na drugu [3].

### 2.4.5. Smoke testiranje

*Smoke* testiranje vrsta je testiranja u kojem se testiraju osnovne funkcionalnosti aplikacije. Važno je pokriti najvažnije značajke i utvrditi da nema grešaka koje uzrokuju nemogućnost korištenja aplikacije. Svaki se novi *deploy* smoke testira kako bi se potvrdilo ima li smisla ići na detaljnije testiranje i rade li kritične funkcionalnosti. Ukoliko se radi nova funkcionalnost, ona se neće moći testirati ako je u međuvremenu pokvarena neka od osnovnih funkcionalnosti zadužena za pokretanje i rad aplikacije općenito. Na primjer, ukoliko se na aplikaciji za internet bankarstvo radi nova funkcionalnost uplate na račun, prvo se mora testirati funkcionalnost prijave i pristupa osnovnom računu kako bi se nova funkcionalnost uopće mogla koristiti [6].




Slika 2.1. *Ciklus smoke testiranja.*

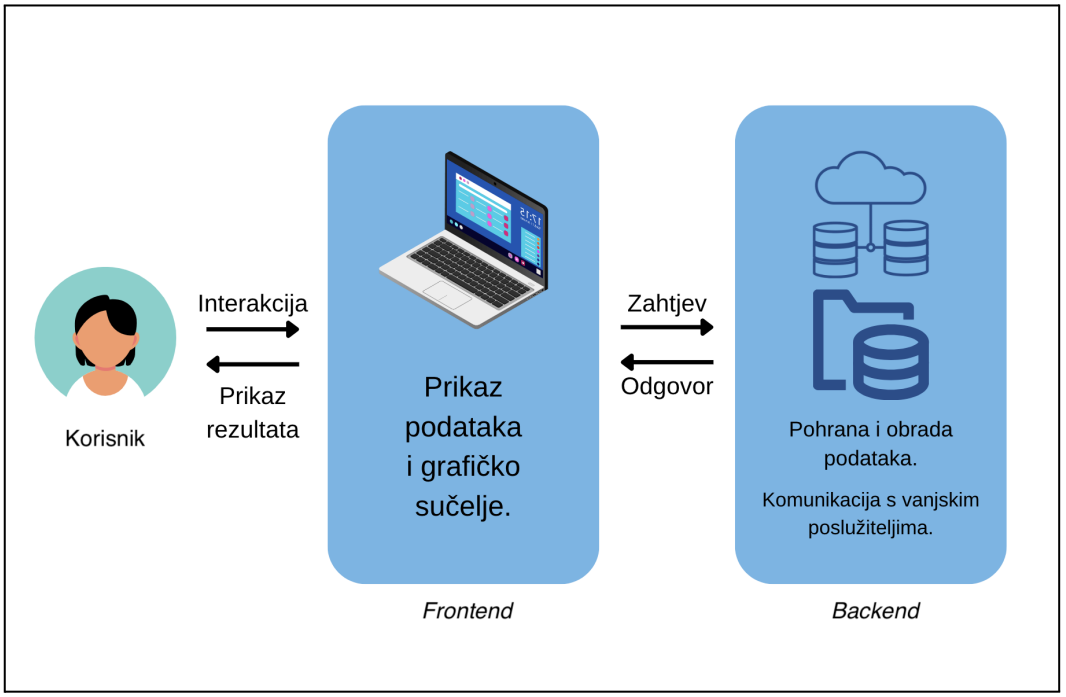
Na slici 2.1. prikazan je ciklus *smoke* testiranja. Testna instanca aplikacije (engl. *build*) dolazi do inženjera osiguranja kvalitete koji izrađuje skup testnih slučajeva kojima pokriva glavne funkcionalnosti aplikacije. Ukoliko je testiranje otkrilo pogreške u radu, aplikacija se vraća razvojnom timu zajedno s izvještajem testiranja te razvojni tim radi na otklanjanju pogrešaka, radi novu instancu aplikacije te ciklus kreće iznova. Kada su testovi prolazni i zadovoljeni, aplikacija se može proslijediti dalje na funkcionalno testiranje.

## 2.5. Testiranje internet aplikacija

Prema [7], 5,18 milijardi ljudi koristi internet. U postocima to iznosi 64,6% ukupne svjetske populacije, a preko 50 milijuna ljudi svakodnevno koristi internet aplikacije koje su glavni posrednik interakcije korisnika s internetom. Kako bi se aplikacije korisnicima dostavile kvalitetne i spremne za korištenje, važno ih je prilikom razvoja detaljno testirati te prevenirati isporučivanje aplikacije s greškama.

### 2.5.1. Struktura internet aplikacija

Internet aplikacije sastoje se od nekoliko programskih slojeva koji čine *frontend* i *backend* aplikacije. *Frontend* sloj zadužen je za prikaz podataka i sučelja korisniku, a *backend* sadrži logiku za pohranu i obradu podataka kao i za komunikaciju vanjskim poslužiteljima (Slika 2.2.).




Slika 2.2. *Arhitektura internet aplikacije.*

Internet aplikacije mogu imati različite strukture, a trenutno najpopularnije su:

1. *Single-Page* struktura internet aplikacije odnosi se na aplikaciju prikazanu u jednom modelu dokumenta koji sadrži sve funkcionalnosti,
2. *Multiple Page* struktura internet aplikacije odnosi se na aplikaciju koja je podijeljena na nekoliko stranica kojima se može pristupiti putem poveznice,

3. *Progressive* struktura internet aplikacije odnosi se na korištenje WebView pružatelja za mobilne uređaje koji omogućava pokretanje internet aplikacije poput nativne mobilne aplikacije [8].

### 2.5.2. Pristup testiranju internet aplikacija

Internet aplikacije pokreću se na internet preglednicima koji se pokreću na operativnom sustavu uređaja. Osim glavnih i osnovnih funkcionalnosti aplikacije, s obzirom na broj slojeva koji može utjecati na performanse aplikacije, važno je testirati ponašanje aplikacije na različitim preglednicima i različitim uređajima kako bi se utvrdilo postoji li odstupanje u ponašanju ovisno o načinu pristupa [9]. Većina preglednika ima ugrađenu alatnu traku s alatima za programere pomoću koje se može pratiti ponašanje aplikacije pod različitim okolnostima. Kako je internet dostupan na gotovo svim postojećim elektroničkim uređajima sa zaslonom, veličina i razlučivost zaslona u različitim kombinacijama također treba biti testirana kako bi se aplikacija namijenjena za korištenje na različitim uređajima zaista i mogla koristiti na njima.

*Backend* se može testirati odvojeno od *frontend* sloja, no u tom slučaju bez korisničkog sučelja, već isključivo izravnim komuniciranjem s poslužiteljem. Temeljitim testiranjem *backend-a* stvara se čvrst temelj na koji se nadovezuje *frontend*. *Frontend* pruža iskustvo testiranja iz korisničke perspektive, dok se testiranjem *backend-a* osim funkcionalnosti, testira i robusnost, sigurnost i različite druge nefunkcionalne značajke [8].

## 2.6. Razlike između manualnog i automatiziranog testiranja

Manualno testiranje vrlo je fleksibilno te se brzo može izvesti i vidjeti rezultate, a za projekte koji su kratkog trajanja nije isplativo postavljati kompleksnu strukturu automatiziranih testova [10]. Manualno testiranje jedini je način za testiranje korisničkog iskustva i ponašanja aplikacije iz perspektive korisnika. Testiranje funkcionalnosti koje zahtijevaju ponavljajuće radnje može postati zamorno pa se detalji mogu propustiti, čemu se može doskočiti automatiziranjem takvih testnih slučajeva. Automatizirano testiranje veoma je korisno i kod zahtjevnih algoritamskih testnih slučajeva. Iako zahtijevaju više resursa pri postavljanju, automatizirani testovi uvelike olakšavaju regresijsko testiranje, testiranje aplikacija u održavanju kao i razne vrste nefunkcionalnog testiranja. U integraciji s kontinuiranim pokretanjem uz svaku izmjenu kôda, osigurava se da su postojeće funkcionalnosti neprestano pokrivene testovima te se smanjuje napor manualnog testiranja postojećih funkcionalnosti [10].

Ne postoji jedinstvena determinanta po kojoj se može odrediti je li bolje manualno ili automatizirano testiranje. Automatizirano testiranje ne može zamijeniti manualno budući da se korisničko iskustvo, uporabljivost, estetika i neke funkcionalnosti ne mogu automatizirati i valjano analizirati rezultate [11].

## 2.7. Alati za automatizirano testiranje

Potreba za različitim načinima testiranja internet aplikacija dovela je do velikog broja alata za automatizirano testiranje. U ovom poglavlju opisani su trenutno najpopularniji alati za pisanje i izvođenje automatiziranih testova. U znanstvenom radu *Automated Software Testing Tools* [12] opisani su najkorišteniji alati za provođenje testiranja raznih nefunkcionalnih značajki poput Apache JMeter-a i LoadRunner-a kao i alati za testiranje funkcionalnih značajki poput Seleniuma.

### 2.7.1. Selenium

Selenium je projekt koji podržava razne alate i biblioteke koji omogućavaju automatizaciju testiranja internet preglednika. Iako se često smatra alatom, Selenium je zapravo razvojni okvir (engl. *framework*) otvorenog kôda koji pruža dodatke i proširenja za oponašanje interakcije korisnika s preglednikom. Korijen Seleniuma jest WebDriver, sučelje u kojem se testovi pišu u obliku naredbi koje će se izvoditi u pregledniku. WebDriver za izvođenje testnih naredbi pokreće preglednik koristeći Selenium poslužitelj te oponaša nativno ponašanje korisnika u interakciji s preglednikom [13].

### 2.7.2. Cypress

Cypress je alat za testiranje internet aplikacija koji podržava *end-to-end* testiranje, testiranje komponenti, integracijsko testiranje i unit testiranje unutar preglednika. Osim virtualnog pokretanja testova u oblaku, Cypress ima mogućnost pokretanja i praćenja testova u stvarnom vremenu i s grafičkim sučeljem gdje se svaka linija izvedenog kôda pretoči u interakciju s preglednikom [14].

### 2.7.3. Playwright

Playwright je Microsoftov alat za automatiziranje testova za internet aplikacije. Omogućava pisanje *end-to-end* testova na različitim internet preglednicima te operacijskim sustavima na kojima se preglednici izvode, a isto tako pokriva i automatizaciju testiranja internet aplikacija na mobilnim uređajima. Za svaki se test kreira kontekst preglednika, odnosno stvara se nova

instanca preglednika prilikom pokretanja testa. Raznolikost podrške omogućava testiranje internet aplikacija na različitim kombinacijama preglednika i uređaja što dodatno doprinosi osiguravanju kompatibilnosti aplikacije [15].

# 3. AUTOMATIZIRANO TESTIRANJE INTERNET APLIKACIJA

Automatizirani testovi za internet aplikacije uključuju korištenje posebnih alata kako bi se testovi izvodili automatizirano, bez da ih inženjeri osiguranja kvalitete izvode ručno. Automatizacija poboljšava objektivnost i korektnost testiranja jer ne dolazi do zamora i testiranje se provodi uvijek na isti način.

## 3.1. Pristup automatiziranom testiranju internet aplikacija

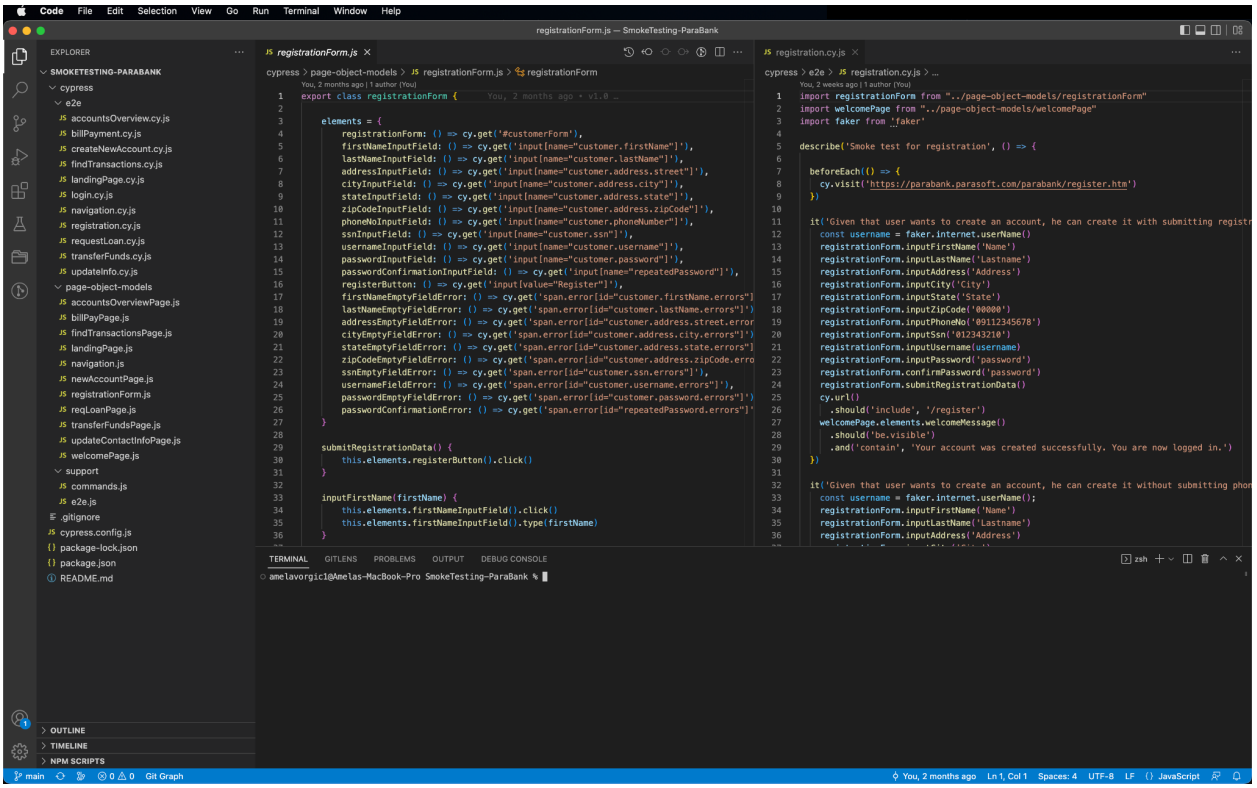
Za automatizaciju testiranja internet aplikacija potrebno je posebno pripremiti skup testnih slučajeva koji su prilagođeni tako da se pomoću napisanog kôda test može izvesti te analizirati [9]. Vrlo je važno razmišljati o tome da se dobiveni rezultat treba vrednovati te prema tome oblikovati test. Potrebno je i odlučiti se za alat koji će se koristiti za pisanje i izvođenje testova, a to ovisi o aplikaciji na kojoj se testiranje izvodi te alatima koji podržavaju testiranje određenih funkcionalnosti na tražen način.

## 3.2. Priprema okruženja za pisanje i izvođenje automatiziranih testova

Kôd programa za izvođenje automatiziranih testova piše se u odabranom uređivaču kôda, a u ovom radu odabrani uređivač kôda koji je ujedno i razvojno okruženje jest Visual Studio Code. kôd je pisan JavaScript programskim jezikom u Cypressu, uz korištenje biblioteke Faker u testovima u kojima je potrebno simulirati podatke.

### 3.2.1. Visual Studio Code

Visual Studio Code jest desktop aplikacija koja se može pokretati na Windows, macOS i Linux operativnim sustavima. Podržava jezike JavaScript, TypeScript i Node.js, ali ima mnoštvo ekstenzija za podršku jezika poput C++, C#, Java, Python i drugih. Visual Studio Code ima intuitivno grafičko sučelje te nudi mogućnost *deploya* kôda na razne platforme poput Githuba i Azurea [17].



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

Na slici 3.1. vidljivo je sučelje Visual Studio Code-a gdje se na lijevoj strani zaslona nalazi traka aktivnosti koja nudi pristup stablu projekta, pretraživanju projekta, kontroli toka i izvora, pokretanju i praćenju izvođenja, dodacima i ekstenzijama, testiranju, projekt menadžeru i ostalim dodacima koje korisnik uveze u projekt. U stablu projekta vidljivi su direktoriji i datoteke u njima, a glavni zaslon prikazuje uređivač kôda te ispod njega terminal pokrenut iz Visual Studio Code-a. Na dnu zaslona nalazi se statusna traka, a na vrhu traka izbornika.

### 3.2.2. JavaScript

JavaScript je jedan od temeljnih programskih jezika za razvoj *World Wide Weba*, zajedno s HTML-om i CSS-om. JavaScript je visokorazinski skriptni jezik koji se izvodi u pregledniku te je prototipno orijentiran. Ima aplikacijsko programsko sučelje *(engl. API)* za rad s tekstom, datumima, standardnim strukturama podataka, regularnim izrazima i DOM *(engl. Document Object Model)*. DOM povezuje internet stranicu sa skriptama ili programskim jezicima u stablastoj strukturi. [18].

### 3.2.3. Faker

Faker je biblioteka koja generira lažne podatke u različitim strukturama. Za potrebe automatiziranog testiranja veoma je povoljno koristiti Faker biblioteku kako bi se ispravan oblik podataka slao dinamično i s različitim vrijednostima svaki puta. Za korištenje Fakera potrebno ga je instalirati kao *dev dependency* i uvesti u projekt. Neki od oblika podataka koje Faker može generirati su: ime, prezime, email, broj telefona, ulica, grad, poštanski broj, žanr glazbe, datum ili različiti oblici ID-a [19].

```
import faker from 'faker'

describe('Smoke test for registration', () => {

  beforeEach(() => {
    cy.visit('https://parabank.parasoft.com/parabank/register.htm')
  })

  it('Given that user wants to create an account, he can create it with submitting registration form', () => {
    const username = faker.internet.userName()
```

Slika 3.2. *Prikaz uvoza Faker biblioteke i primjene na kreiranju nasumične vrijednosti za varijablu „username”.*

Na slici 3.2. vidljiv je kôd u kojem je u datoteci *registration.cy.js* Faker uvezen te korišten. Prilikom deklaracije varijable *username* dodijeljena joj je nasumična vrijednost oblika korisničkog imena koju prilikom izvođenja generira biblioteka Faker. Na slici 3.3. prikazano je korištenje biblioteke Faker za nasumično generiranje podataka oblika: ime, prezime, ulica, grad država, broj i broj telefona. Ovaj kôd korišten je za kreiranje nasumičnih podataka prilikom testiranja funkcionalnosti ažuriranja podataka.

```
    const firstName = faker.name.firstName()
    const lastName = faker.name.lastName()
    const address = faker.address.streetName()
    const city = faker.address.city()
    const state = faker.address.state()
    const zipCode = faker.random.number()
    const phone = faker.phone.phoneNumber()
```

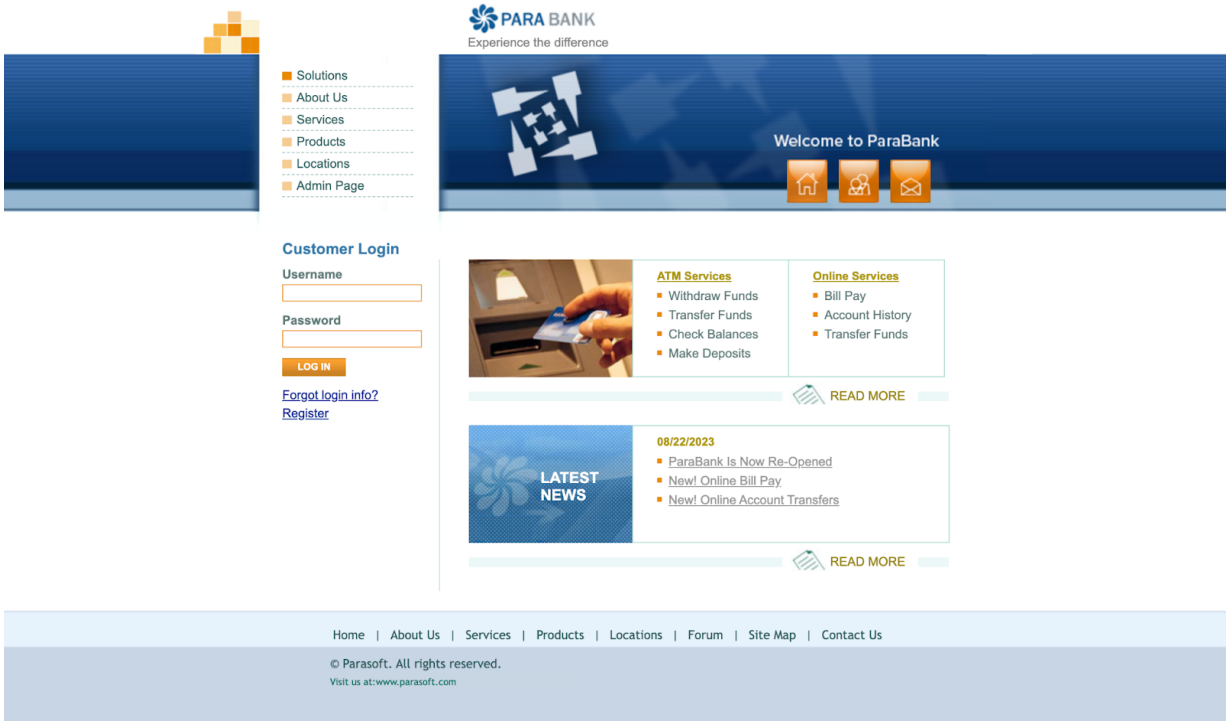
Slika 3.3. *Korištenje Faker biblioteke za kreiranje nasumičnih vrijednosti različitog oblika.*

# 4. IMPLEMENTACIJA AUTOMATIZIRANIH TESTOVA

U ovom poglavlju opisana je internet aplikacija ParaBank na kojoj se testiranje izvodi kao i način pripreme projekta i modela objekata stranica pomoću kojih se elementi na stranici dohvaćaju za izvođenje testova. Opisani su i testni slučajevi koji će se automatizirati kao i proces izvođenja automatiziranih testova.

## 4.1. ParaBank internet aplikacija

ParaBank je demo internet aplikacija tvrtke Parasoft. Funkcionalnosti koje ima simuliraju aplikaciju koja pruža usluge internet bankarstva [20]. Internet aplikaciji se može pristupiti na poveznici https://parabank.parasoft.com/parabank/index.htm.




Slika 4.1. *Početno sučelje aplikacije ParaBank.*

Na slici 4.1. prikazano je početno sučelje aplikacije ParaBank te se mogu vidjeti neke funkcionalnosti koje nudi.

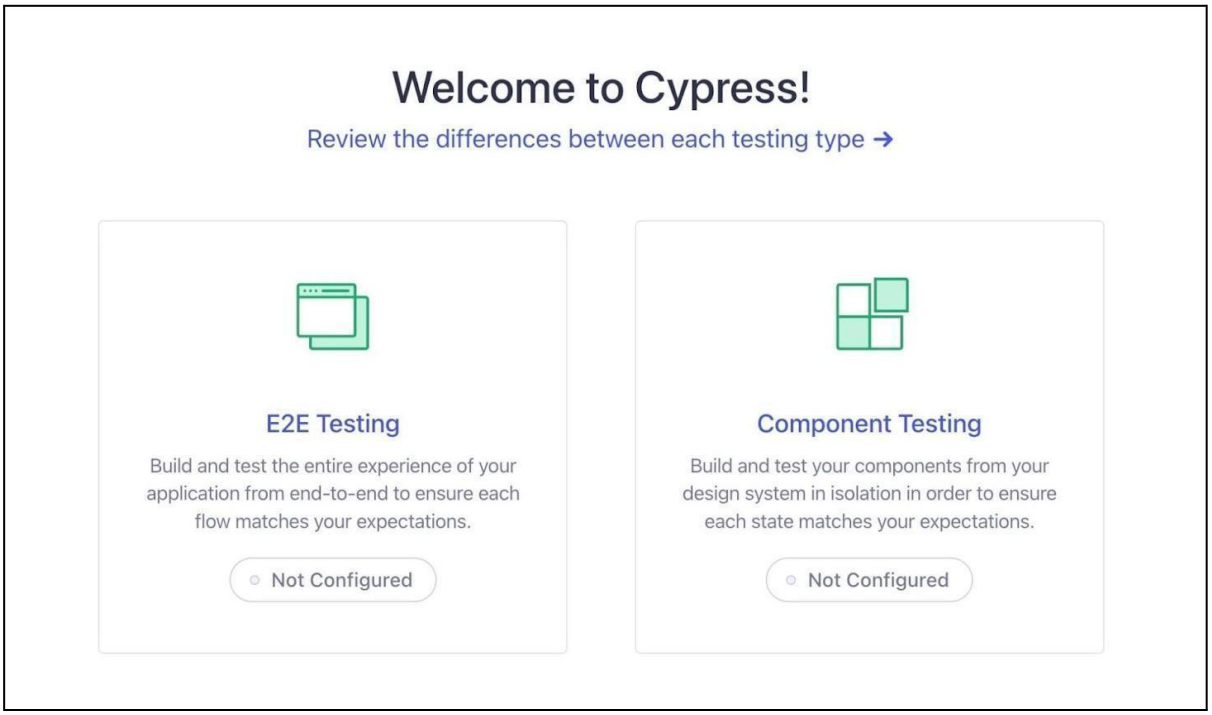
Lista svih funkcionalnosti koje aplikacija ima te koje se testiraju:

- registracija,
- prijava,
- otvaranje novog računa,

- pregled računa,
- prijenos sredstava plaćanje računa,
- pronalazak transakcija,
- ažuriranje podataka,
- zahtijevanje pozajmice.

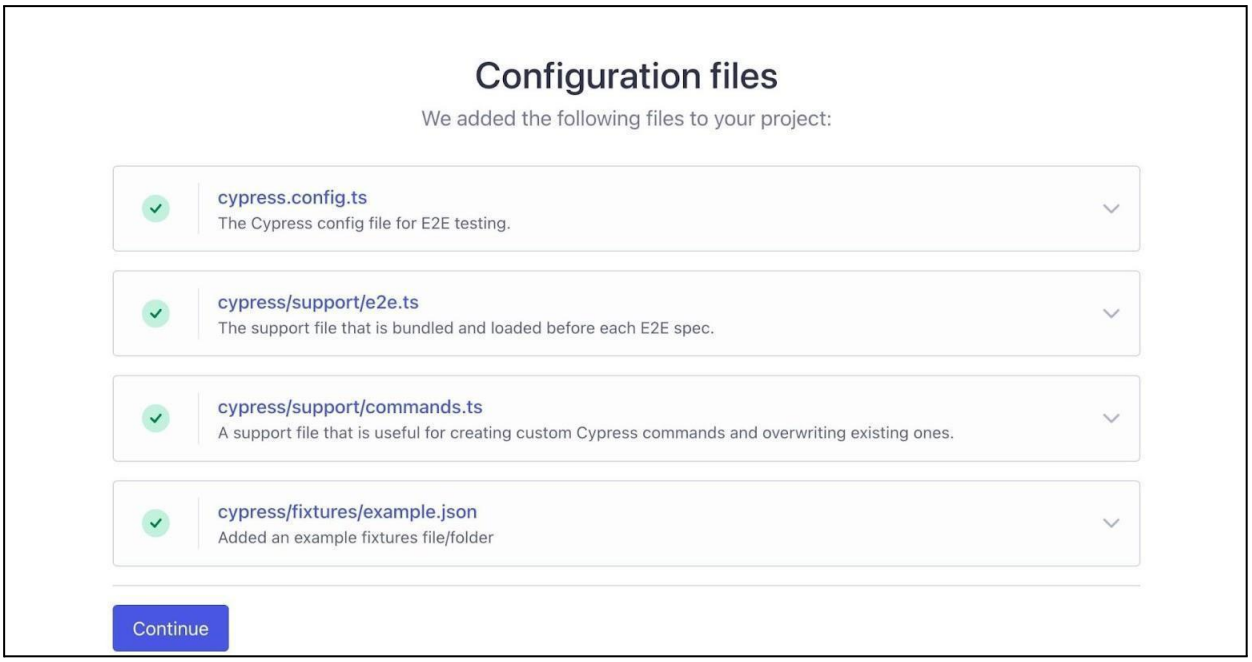
## 4.2. Instalacija Cypressa

Kako bi se Cypress mogao koristiti za proces automatizacije testova, potrebno ga je instalirati u projekt. Nakon kreiranja novog projekta potrebno je unutar projekta otvoriti terminal te pokrenuti naredbu `npm install cypress --save-dev`. Po instaliranju Cypressa, potrebno ga je pokrenuti naredbom `npx cypress open`. Tada će se otvoriti sučelje sa slike 4.2.



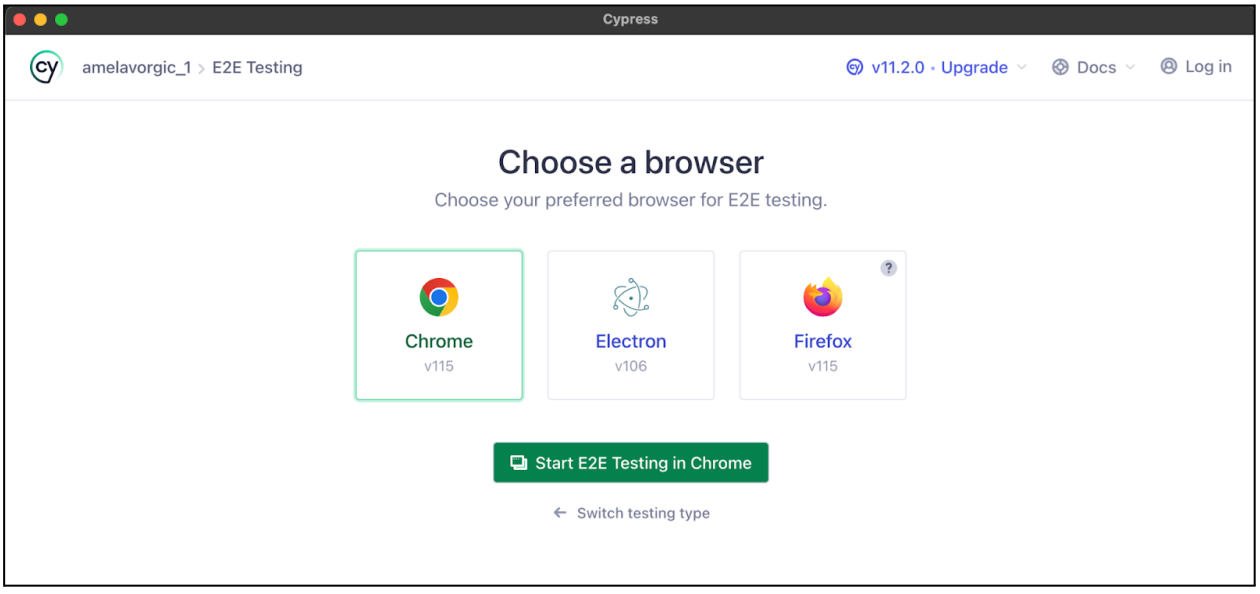

Slika 4.2. *Sučelje za konfiguraciju Cypressa.*

Potrebno je odabrati *E2E Testing* gdje će se prikazati osnovne datoteke potrebne za korištenje Cypressa koje će se automatski generirati (Slika 4.3.).




Slika 4.3. *Konfiguracijske datoteke.*

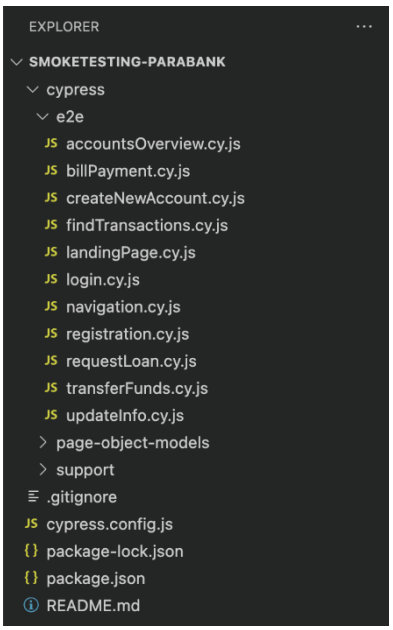
Pritiskom na gumb *Continue* otvara se sučelje za odabir preglednika u kojem će se testovi izvoditi (Slika 4.4.)




Slika 4.4. *Sučelje za odabir preglednika.*

Izbor preglednika ovisi o računalu na kojemu se projekt pokreće i koji su preglednici na njemu već instalirani. Kada se pokrene testiranje na odabranom pregledniku, Cypress nudi kreiranje predloška *spec* datoteke u kojoj se testovi pišu. Svi testovi moraju se nalaziti u datoteci s ekstenzijom .cy kako bi ih Cypress prepoznao kao testove. *Spec* datoteke uglavnom se nalaze u

direktoriju *e2e*. Na slici 4.5. prikazano je stablo projekta s izlistanim *spec* datotekama u kojima se nalaze testovi.




Slika 4.5. *Stablo projekta s izlistanim spec datotekama.*

## 4.3. Spec datoteka

*Spec* datoteka se sastoji od nekoliko dijelova s različitim funkcionalnostima. Dobra je praksa imenovati datoteku prema funkcionalnosti koja se testira. *Spec* datoteka ima ekstenziju .cy koja označava da se radi o datoteci u kojoj se testovi nalaze, a ekstenzija .js označava da je kôd pisan programskim jezikom JavaScript. Unutar *spec* datoteke, a prije ostatka kôda, mogu se uvesti (engl. *import*) druge datoteke iz projekta kao i vanjske biblioteke. Na slici 4.6. prikazan je predložak s kojim se svaki testni scenarij može raspisati u obliku kôda.

```
describe('Opis testnog scenarija', () => {
  it('Opis testnog slučaja', () => {
    //koraci za izvođenje testa
  })
})
```

Slika 4.6. *Predložak kôda za pisanje testova.*

Blok *describe*() obuhvaća više *it*() blokova koji se ponašaju kao zasebni testni slučajevi. Unutar metode *describe*() predaje se parametar tipa *string* u kojem se generalno opisuje testni scenarij. Na slici 4.7. prikazan je primjer *describe*() metode iz datoteke *registration*.cy.js.


Slika 4.7. *describe() metoda iz datoteke registration.cy.js.*

U blok *it*() se također predaje parametar tipa *string* u kojem se detaljno opisuje testni slučaj. Na primjeru sa slike 4.8. je vidljivo na koji se način testni slučaj može definirati.

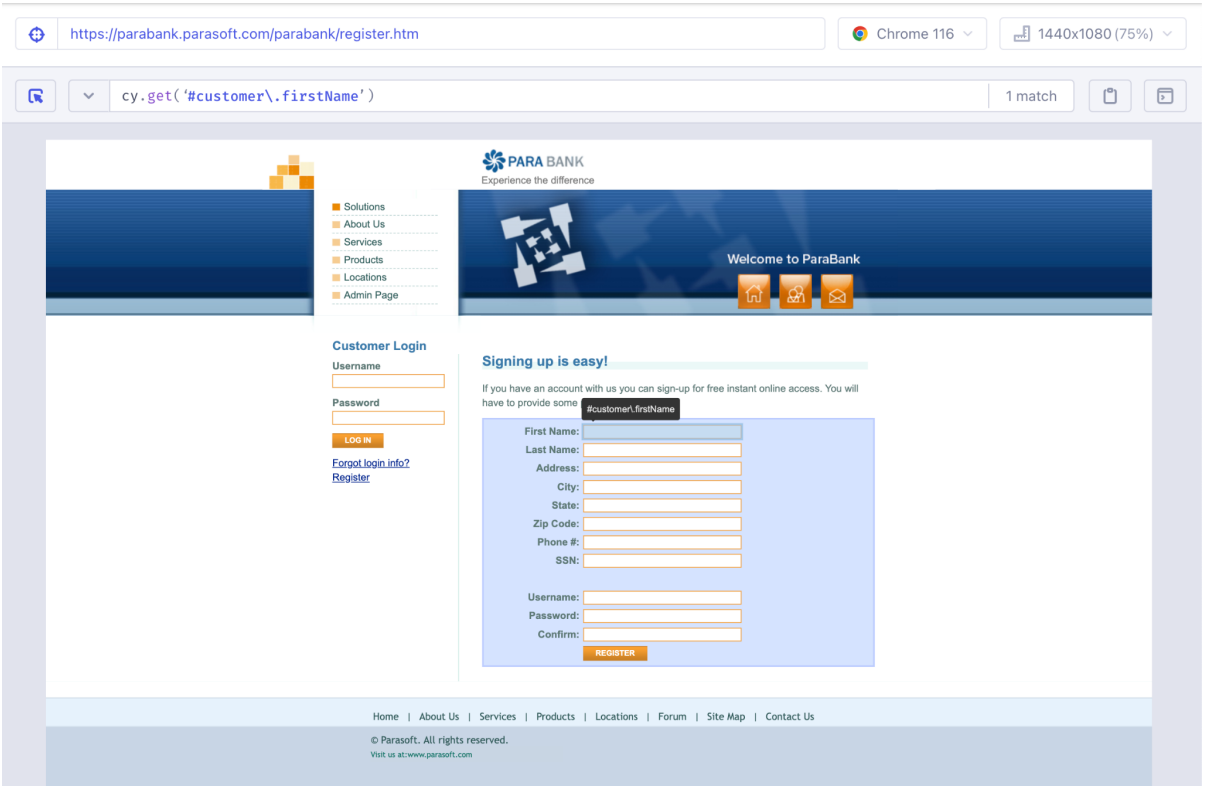

Slika 4.8. *it() metoda iz datoteke registration.cy.js s testnim slučajem kreiranja korisničkog računa s korisničkim imenom koje je već registrirano.*

## 4.4. Page Object Models

Modeli objekata stranice (engl. *Page Object Models*) kreiraju se u obliku klasa stranice koju predstavljaju. Modeli objekata stranice zapravo su repozitoriji za pohranu elemenata sa stranice. U klasi se definiraju elementi koje stranica sadrži, a s kojima se treba izvesti interakcija kako bi se simuliralo ponašanje korisnika. U klasi se također definiraju i metode koje obrađuju podatke na elementima [21]. Uz korištenje modela objekata stranice cijela se internet aplikacija može razložiti na zasebne klase koje predstavljaju pojedinačne stranice. Održavljivost kôda je puno veća korištenjem modela objekata stranice jer se funkcionalni kôd i logika testova ne treba mijenjati ukoliko dođe do promjena u izgledu stranice, već se izmjene rade u klasama za potrebne stranice.

### 4.4.1. Lokatori

Lokatori su jedinstveni objekti koji pronalaze i vraćaju element na internet stranici. Kako bi se testovi mogli izvoditi potrebno je prvo locirati element s kojim se treba komunicirati kako bi se radnja nad njim mogla izvesti. Dobra praksa prilikom izrade bilo kakve aplikacije jest svakom elementu dodijeliti jedinstveni ID koji će služiti za dohvaćanje elementa. Ukoliko element nema ID, uglavnom ima dinamičku putanju pomoću koje se može dohvatiti, ali to se treba izbjegavati budući da se izmjenama kôda može dogoditi i promjena putanje. Osim ID-a i putanje, element se može dohvatiti i s različitim oznakama (engl. *tag*), atributima i klasama kojima pripada [22]. Cypress ima ugrađeni selektor kojim je moguće izravno sa zaslona dohvatiti željeni element. Na slici 4.10. prikazan je rad Cypress selektora te lokator koji je korištenjem njega dobiven.




Slika 4.10. *Korištenje Cypress selektora.*

Ponekad lokatori koje Cypress selektor dohvati nisu u optimalnom formatu, a kako postoji više načina za dohvaćanje moguće je koristiti i lokator drugačijeg formata. Osim Cypress selektora, lokatore je moguće odrediti korištenjem inspektora preglednika i dohvaćanjem CSS-a i HTML-a internet aplikacije. Odabirom elementa korištenjem inspektora, prikazuje se programski kôd kojim je element ostvaren te njegovi atributi. Na slici 4.11. prikazan je element polja za unos imena te kôd kojim je element ostvaren. Iz kôda je moguće izvući tip polja kao i njegov jedinstveni ID po kojem se može locirati na stranici.


Slika 4.11. *Korištenje inspektora preglednika za identificiranje elementa.*

Na slici 4.12. može se vidjeti način na koji su elementi s internet aplikacije ParaBank dohvaćeni. Na ParaBank internetskoj aplikaciji većina elemenata ima jedinstveni ID pa je svaki element

jednoznačno definiran i time su testovi, kao i aplikacija sveukupno, kvalitetniji i otporniji na promjene.

```
3     elements = {
4         registrationForm: () => cy.get('#customerForm'),
5         firstNameInputField: () => cy.get('input[name="customer.firstName"]'),
6         lastNameInputField: () => cy.get('input[name="customer.lastName"]'),
7         addressInputField: () => cy.get('input[name="customer.address.street"]'),
8         cityInputField: () => cy.get('input[name="customer.address.city"]'),
9         stateInputField: () => cy.get('input[name="customer.address.state"]'),
10        zipCodeInputField: () => cy.get('input[name="customer.address.zipCode"]'),
11        phoneNoInputField: () => cy.get('input[name="customer.phoneNumber"]'),
12        ssnInputField: () => cy.get('input[name="customer.ssn"]'),
13        usernameInputField: () => cy.get('input[name="customer.username"]'),
14        passwordInputField: () => cy.get('input[name="customer.password"]'),
15        passwordConfirmationInputField: () => cy.get('input[name="repeatedPassword"]'),
16        registerButton: () => cy.get('input[value="Register"]'),
17        firstNameEmptyFieldError: () => cy.get('span.error[id="customer.firstName.errors"]'),
18        lastNameEmptyFieldError: () => cy.get('span.error[id="customer.lastName.errors"]'),
19        addressEmptyFieldError: () => cy.get('span.error[id="customer.address.street.errors"]'),
20        cityEmptyFieldError: () => cy.get('span.error[id="customer.address.city.errors"]'),
21        stateEmptyFieldError: () => cy.get('span.error[id="customer.address.state.errors"]'),
22        zipCodeEmptyFieldError: () => cy.get('span.error[id="customer.address.zipCode.errors"]'),
23        ssnEmptyFieldError: () => cy.get('span.error[id="customer.ssn.errors"]'),
24        usernameFieldError: () => cy.get('span.error[id="customer.username.errors"]'),
25        passwordEmptyFieldError: () => cy.get('span.error[id="customer.password.errors"]'),
26        passwordConfirmationError: () => cy.get('span.error[id="repeatedPassword.errors"]')
27    }
```

Slika 4.12. *Lokatori na objektu modela stranice za registraciju.*

### 4.4.2. Metode

Kako bi interakcija s elementima bila jednostavnija, a kôd čišći i pregledniji, kreirane su metode za unos podataka u polja za unos kao i metode za predaju podataka klikom na gumb. Na slici 4.13. prikazane su metode *submitRegistrationData(), inputFirstName(), inputLastName()* i *inputAddress()*.

```
29      submitRegistrationData() {
30          this.elements.registerButton().click()
31      }
32
33      inputFirstName(firstName) {
34          this.elements.firstNameInputField().click()
35          this.elements.firstNameInputField().type(firstName)
36      }
37
38      inputLastName(lastName) {
39          this.elements.lastNameInputField().click()
40          this.elements.lastNameInputField().type(lastName)
41      }
42
43      inputAddress(address) {
44          this.elements.addressInputField().click()
45          this.elements.addressInputField().type(address)
46      }
```

Slika 4.13. *Metode za interakciju s dohvaćenim elementima.*

Metoda *submitRegistrationData()* dohvaća gumb za predaju forme, te se pozivom ove metode pritišće gumb na formi za registraciju na jednak način na koji bi korisnik napravio klik mišem. Metoda *inputFirstName()* kao ulazni parametar prima varijablu *firstName*. Kada se metoda pozove u kôdu, potrebno joj je predati *string* tip podatka koji se želi spremiti kao ime. Prvo se izvodi klik na ulazno polje kako bi se osiguralo da se vrijednost predaje te se koristi naredba *type()* koja u to polje predaje ulazni parametar. Metode *inputLastName()* i *inputAddress()* kao i ostale kreirane metode rade po istom principu. Dohvaća se željeni element te se na njega klikne i nakon toga se u polje upisuje vrijednost predanog parametra.

## 4.5. Hooks

*Hook* omogućava određivanje seta uvjeta koji se trebaju izvesti prije ili nakon izvođenja svih ili nekih od testova. Koriste se za postavljanje početnih stanja i pripremu za izvođenje testa kao i za čišćenje podataka nakon izvođenja. *Hook before()* izvodi se jednom, prije svih testova unutar datoteke, dok se *hook beforeEach()* izvodi prije svakog testa unutar datoteke. *Hook after()* izvodi se jednom, po završetku izvođenja svih testova, a *hook afterEach()* izvodi se nakon završetka svakog pojedinog testa.

## 4.6. End-To-End testiranje

*End-to-End* testiranje je tehnika testiranja kojom se testira funkcionalnost i ponašanje čitave aplikacije simuliranjem scenarija ponašanja pravih korisnika i podataka. Cilj ove tehnike jest pronaći greške u radu aplikacije s integriranim komponentama kako bi gotov proizvod radio na način na koji je zamišljen. Inženjer osiguranja kvalitete postavlja se u ulogu korisnika te iz korisničke perspektive izvodi testiranje prema slučajevima koje bi korisnik izvodio [23].

### 4.6.1. Testni slučajevi

Testni slučaj jest zamišljeni scenarij koji oponaša radnju korisnika te uz niz koraka koji se trebaju izvesti ima propisani očekivani rezultat. Izvođenjem testa prema koracima iz testnog slučaja, dobiveni se rezultat uspoređuje s očekivanim te se na taj način validira ponašanje aplikacije.

Za testiranje aplikacije ParaBank raspisani testni slučajevi su podijeljeni u jedanaest kategorija:

- početna stranica,
- registracija,
- prijava,

- navigacija,
- otvaranje novog računa,
- pregled računa,
- prijenos sredstava plaćanje računa,
- pronalazak transakcija,
- ažuriranje podataka,
- zahtijevanje pozajmice.

Testiranje funkcionalnosti registracije pokriveno je s četrnaest testnih slučajeva. Jedan od testnih slučajeva jest „Kao korisnik, želim imati mogućnost kreiranja računa na internet aplikaciji ParaBank predajom podataka u obrascu za registraciju". Koraci koji se trebaju izvesti za testiranje ovog slučaja su:

1. otvori https://parabank.parasoft.com/parabank/register.htm,
2. unesi ime,
3. unesi prezime,
4. unesi adresu,
5. unesi grad,
6. unesi državu,
7. unesi poštanski broj,
8. unesi broj telefona,
9. unesi broj osiguranja,
10. unesi korisničko ime,
11. unesi lozinku,
12. potvrdi lozinku,
13. klikni „Register".

Očekivani rezultat nakon izvođenja ovog testa jest da je korisnički račun uspješno kreiran te da je korisniku prikazana poruka o uspjehu kreiranja računa.

Kôd za izvođenje ovog testa prikazan je na slici 4.14.

```
11  it('Given that user wants to create an account, he can create it with submitting registration form', () => {
12    const username = faker.internet.userName()
13    registrationForm.inputFirstName('Name')
14    registrationForm.inputLastName('Lastname')
15    registrationForm.inputAddress('Address')
16    registrationForm.inputCity('City')
17    registrationForm.inputState('State')
18    registrationForm.inputZipCode('00000')
19    registrationForm.inputPhoneNo('09112345678')
20    registrationForm.inputSsn('012343210')
21    registrationForm.inputUsername(username)
22    registrationForm.inputPassword('password')
23    registrationForm.confirmPassword('password')
24    registrationForm.submitRegistrationData()
25    cy.url()
26      .should('include', '/register')
27    welcomePage.elements.welcomeMessage()
28      .should('be.visible')
29      .and('contain', 'Your account was created successfully. You are now logged in.')
30  })
```

Slika 4.14. *Kôd za testni slučaj „Kao korisnik, želim imati mogućnost kreiranja računa na internet aplikaciji ParaBank predajom podataka u obrascu za registraciju“.*

U varijablu *username* sprema se nasumično generirani string formata *username* koji će se kasnije u kôdu koristiti za odabir korisničkog imena. Biblioteka Faker koristi se kako bi se izbjeglo *hard-coding* vrijednosti. Prateći raspisani testni slučaj, potrebno je unijeti: ime, prezime, adresu, grad, državu, poštanski broj, broj telefona, broj osiguranja, korisničko ime, lozinku, potvrditi lozinku te kliknuti na gumb Register.

## 4.7. Evaluacija rezultata

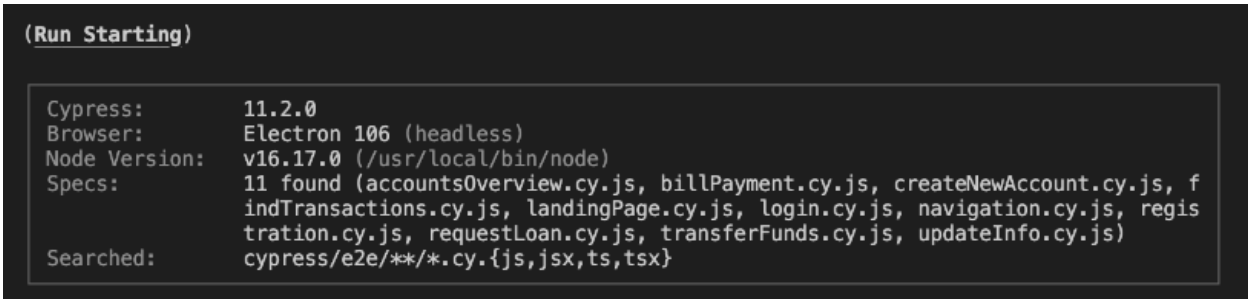
Po završetku izvođenja koraka u izvođenju testa, krajnji rezultat je potrebno evaluirati. Evaluacija rezultata provodi se korištenjem *assertion*-a. *Assertion* uspoređuje dobiveni rezultat s očekivanim rezultatom i na temelju toga testovi prolaze ili padaju. *Assertion* u Cypressu koristi popularnu *assertion* biblioteku Chai. Chai biblioteka omogućuje pisanje *assertion* naredbi nalik na rečenice te se dobiveni rezultat uspoređuje sa zadanim. Očekivani rezultat piše se u obliku „treba” (engl. *should*). Dobiveni rezultat se s očekivanim uspoređuje prema tome sadržava li element određeni string, prikazuje li se određeni element na stranici, odgovara li URL zadanom, jesu li gumbi omogućeni nakon određene akcije i slično [24].

## 4.8. Izvođenje automatiziranih testova

Kada je napisan programski kôd za automatizaciju testova, potrebno ga je pokrenuti kako bi se testovi počeli izvoditi. U Cypressu je moguće testove pokrenuti bez otvaranja testnog preglednika (engl. *headless*) i s otvaranjem testnog preglednika.

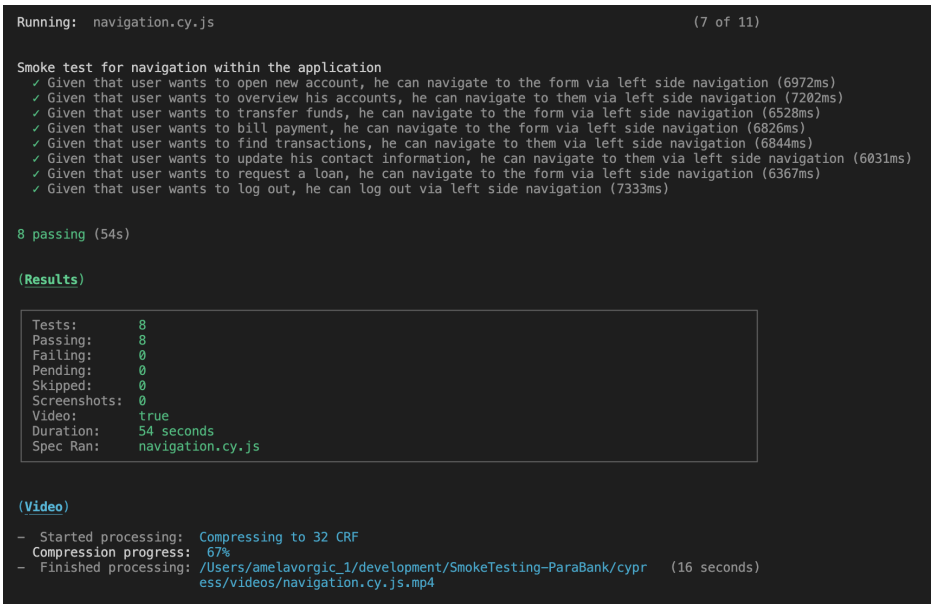
### 4.8.1. Headless pokretanje

Za pokretanje testova i izvođenje bez pokretanja preglednika potrebno je unutar terminala pokrenuti naredbu `npx cypress run`. Ova naredba pokrenut će izvođenje testova s prikazom rezultata unutar terminala (Slika 4.15).



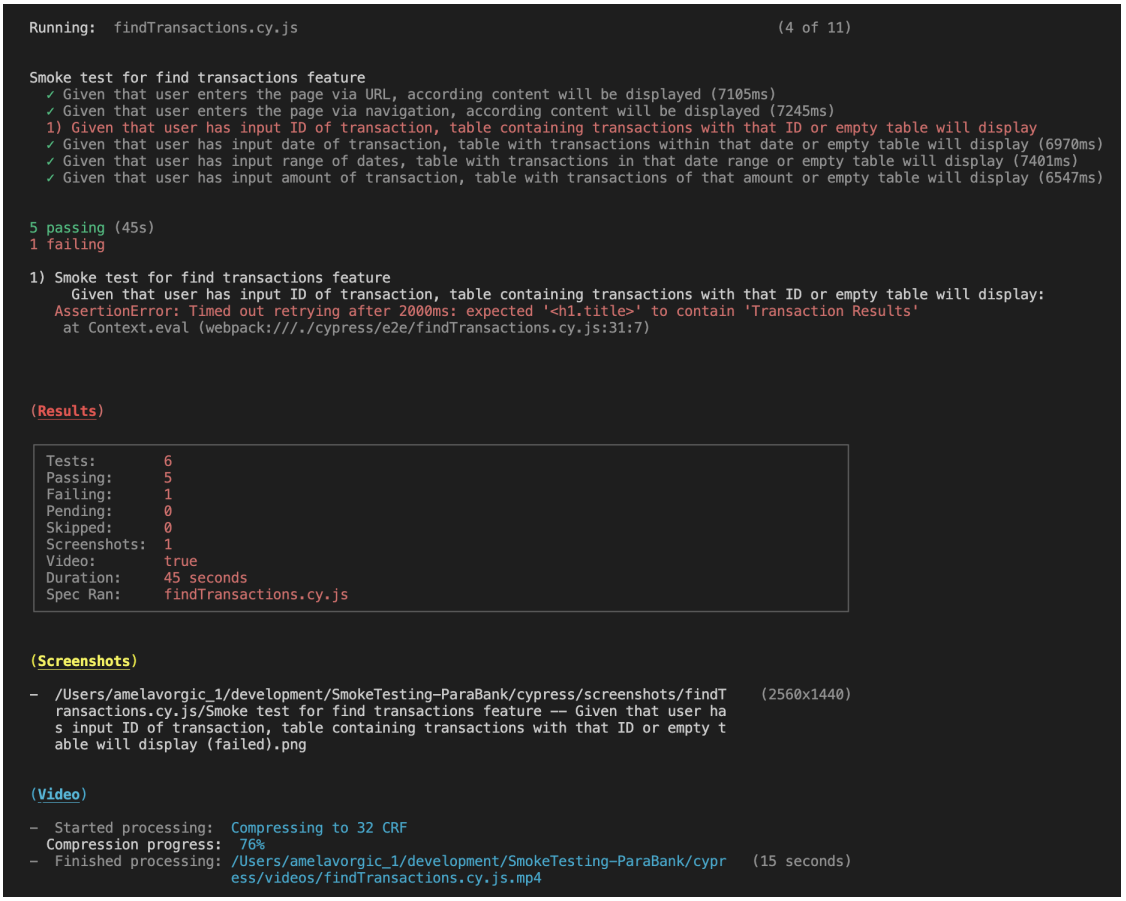

Slika 4.15. *Pokretanje testova bez preglednika.*

Na slici 4.15. može se vidjeti da se prilikom pokretanja testova ispisuje verzija Cypressa kao i verzija preglednika na kojem će se testovi izvoditi na udaljenom serveru. Također su prikazane i pronađene *spec* datoteke koje sadrže testove. Testovi se izvode jedan po jedan, redom po pronađenim *spec* datotekama. Na slici 4.16. prikazan je rezultat izvođenja testova unutar datoteke navigation.cy.js u kojoj se nalaze testovi za testiranje funkcionalnosti navigacije unutar aplikacije putem navigacijske trake. Uz detaljan pregled testnih slučajeva unutar datoteke, ispisuje se i tablica s rezultatima i ukupnim trajanjem izvođenja. Video izvođenja testova sprema se u posebnu datoteku te se može pogledati naknadno.




Slika 4.16. *Prikaz rezultata uspješno izvedenih testova.*

Ukoliko test padne jer se *assert* nije uspješno dogodio i krajnji rezultat izvođenja kôda nije očekivani, detaljniji pregled uz pogrešku će biti prikazan. Na slici 4.17. prikazana je analiza izvođenja testova za funkcionalnost pronalaska transakcija gdje je vidljivo da je jedan test pao (engl. *failed*).

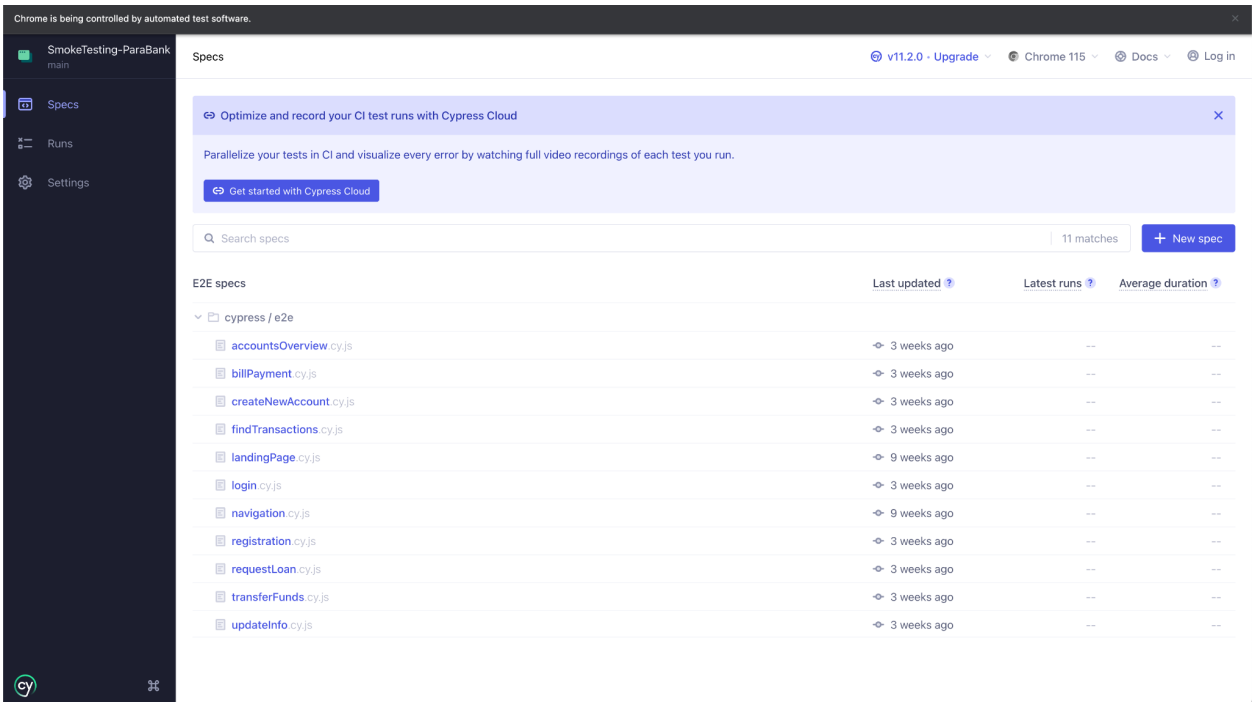



Slika 4.17. *Prikaz rezultata u slučaju pada testa.*

Uz uobičajeni prikaz i analizu izvođenja testova, prikazana je i pogreška zbog koje je test pao, a u ovom slučaju to je pogreška u kojoj se nije prikazala tablica s rezultatima na stranici, koja se identificirala pomoću naslova tablice „*Transactions results*". Osim videa izvođenja, u slučaju kada test padne, spremaju se i snimke zaslona (engl. *screenshots*) s detaljnim pregledom koraka izvođenja.

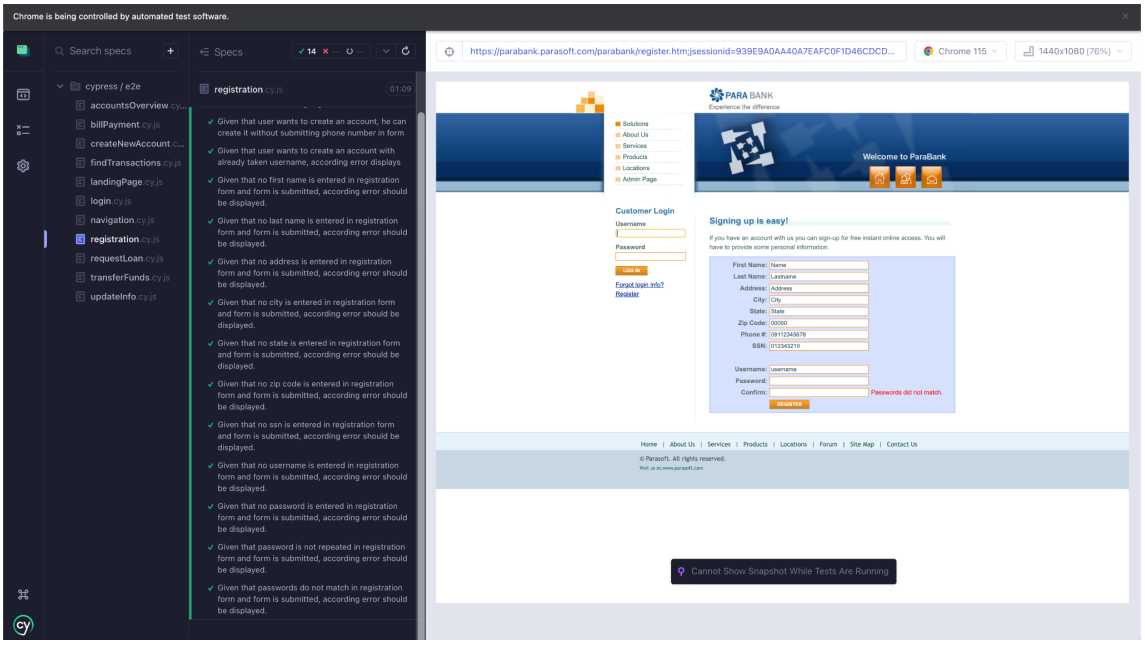
### 4.8.2. Pokretanje unutar preglednika

Pokretanje testova unutar testnog preglednika pokreće se naredbom `npx cypress open`. Izvođenjem ove naredbe automatski se pokreće preglednik u kojem se može pratiti izvođenje testova. Na slici 4.18. prikazane su sve pronađene *spec* datoteke koje se mogu pokrenuti za izvođenje testova.




Slika 4.18. *Prikaz pronađenih spec datoteka u testnom pregledniku.*

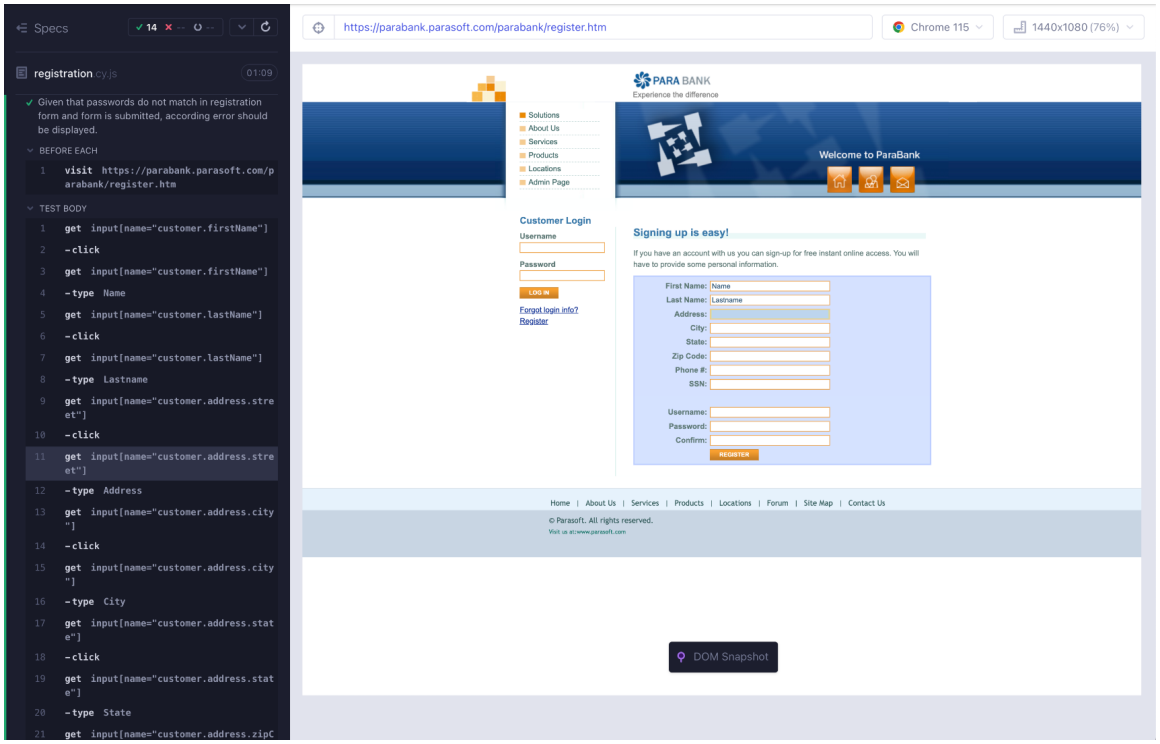
Testovi se pokreću klikom na datoteku. Testovi se izvode jedan po jedan, a svaki se korak uživo (engl. *real-time*) prati na prikazu preglednika (Slika 4.19).




Slika 4.19. *Praćenje izvođenja testova u testnom pregledniku.*

Klikom na određeni testni slučaj, otvaraju se koraci za izvođenje te je moguće klikom na svaki od koraka vidjeti što se točno u tom trenutku događalo na stranici. Na slici 4.20. vidljivo je da se

u jedanaestoj liniji kôda dohvaća polje za unos adrese, a na stranici unutar testnog preglednika vidljivo je da je odabrano polje za unos adrese




Slika 4.20. *Pregled koraka u izvođenju testa.*

# 5. ANALIZA REZULTATA

U ovom se poglavlju analiziraju rezultati provođenja automatiziranih testova te se uspoređuju s rezultatima provođenja manualnih testova.

## 5.1. Analiza rezultata automatiziranih testova

Nakon *headless* izvođenja testova ispisuje se tablica s ukupnim rezultatima prikazana na slici 5. 1.

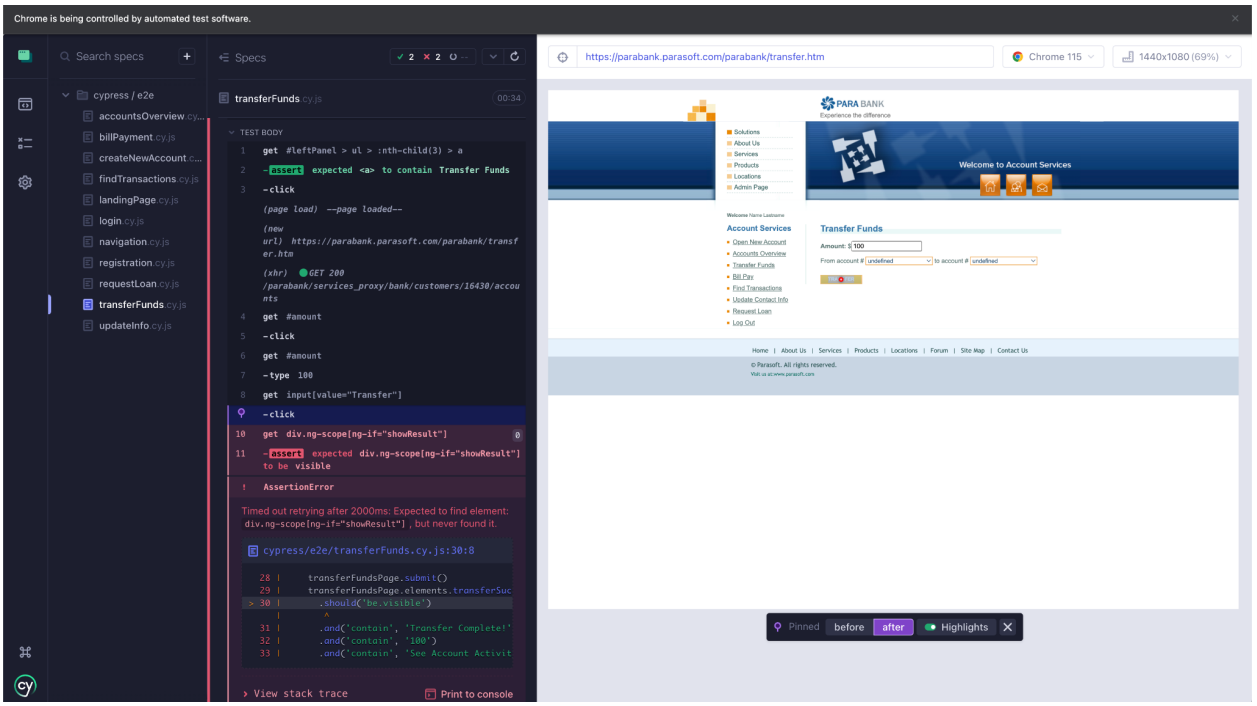
(Run Finished)

| | Spec | | Tests | Passing | Failing | Pending | Skipped |
|---|---|---|---|---|---|---|---|
| ✔ | accountsOverview.cy.js | 00:16 | 2 | 2 | – | – | – |
| ✔ | billPayment.cy.js | 00:24 | 3 | 3 | – | – | – |
| ✔ | createNewAccount.cy.js | 00:27 | 4 | 4 | – | – | – |
| ✖ | findTransactions.cy.js | 00:45 | 6 | 5 | 1 | – | – |
| ✔ | landingPage.cy.js | 00:05 | 3 | 3 | – | – | – |
| ✔ | login.cy.js | 00:06 | 1 | 1 | – | – | – |
| ✔ | navigation.cy.js | 00:54 | 8 | 8 | – | – | – |
| ✔ | registration.cy.js | 01:03 | 14 | 14 | – | – | – |
| ✔ | requestLoan.cy.js | 00:18 | 3 | 3 | – | – | – |
| ✖ | transferFunds.cy.js | 00:31 | 4 | 2 | 2 | – | – |
| ✔ | updateInfo.cy.js | 01:25 | 10 | 10 | – | – | – |
| ✖ | 2 of 11 failed (18%) | 06:19 | 58 | 55 | 3 | – | – |

Slika 5.1. *Analiza rezultata automatiziranih testova.*

Budući da se usporedba očekivanih i stvarnih rezultata odvija prilikom izvođenja testova (*assert*), nije potrebno dodatno testirati svaki slučaj. Ukoliko dođe do pada testa, potrebno je pregledati izvješće o odvijanju izvođenja testa te priložene snimke zaslona. Prema snimkama zaslona moguće je odrediti u kojem je koraku testa došlo do pogreške te se treba napisati izvješće o pogrešci u radu aplikacije. Izvješće o pogrešci treba sadržavati naslov iz kojega je jasno o kakvoj je pogrešci riječ, detalje o pregledniku i operativnom sustavu na kojem se pogreška događa, korake kojima se greška može reproducirati te očekivani i stvarni rezultat.

Automatiziranim testiranjem pronađene su tri pogreške u radu aplikacije, jedna u funkcionalnosti pronalaska transakcije gdje se prilikom pretraživanja transakcija po ID-u prikazuje greška, a dvije u funkcionalnosti prijenosa sredstava gdje se prilikom pokretanja aplikacije u testnom pregledniku broj računa prikazuje kao „*undefined*" i ne može se promijeniti (Slika 5.2) te se test ne može ispravno izvesti.




Slika 5.2. *Pogreška u funkcionalnosti prijenosa sredstava.*

Na slici 5.1. je vidljivo da je za izvođenje sveukupno 58 testnih slučajeva podijeljenih u 11 *spec* datoteka bilo potrebno 6:19 minuta. Uzme li se prosjek, za izvođenje jednog testnog slučaja potrebno je 6,5 sekundi. Od 58 izvedenih testova 55 je bilo uspješno, a 3 testa su pala. Uspješnost testova iznosi 95%. Za pisanje kôda automatiziranih testova utrošeno je 25 radnih sati.

## 5.2. Analiza rezultata manualnih testova

Provođenje manualnih testova izvodilo se praćenjem raspisanih testnih slučajeva iz Priloga 2., a prikazano je u Tablici 1.

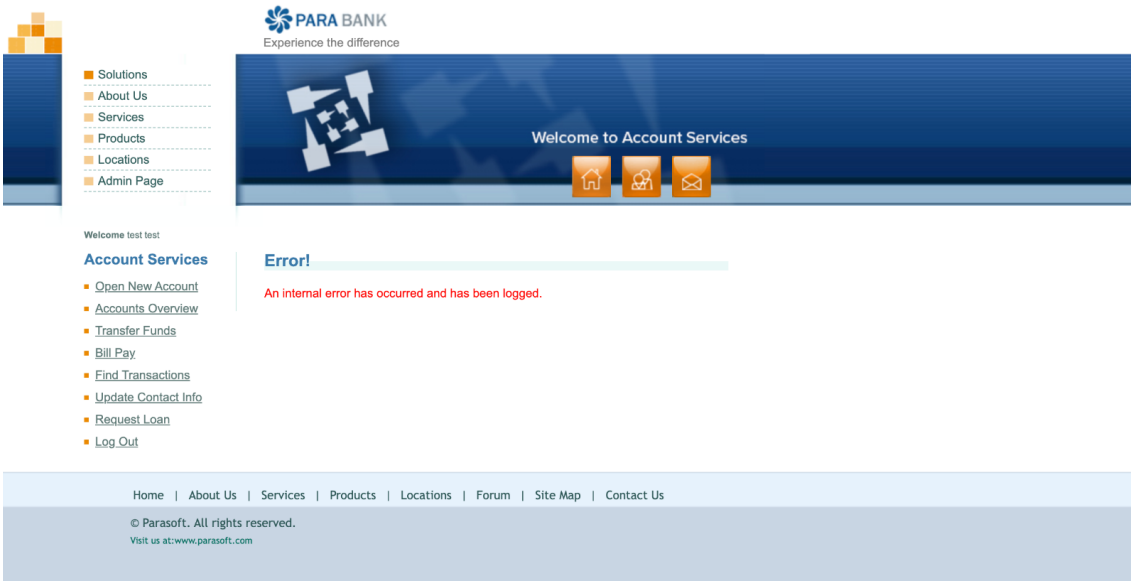
Tablica 1. *Analiza izvođenja manualnog testiranja.*

| | |
|---|---|
| **Broj testova** | 58 |
| **Izvedeni uspješno** | 57 |
| **Izvedeni neuspješno** | 1 |
| **Vrijeme** | 50 minuta |

Prilikom manualnog testiranja također je otkrivena pogreška u radu aplikacije u funkcionalnosti pronalaska transakcija po ID-u. Očekivano ponašanje ukoliko ne postoji niti jedna transakcija s pretraženim ID-om jest da se prikaže tablica bez rezultata, a stvarno ponašanje jest takvo da se prikaže greška bez dodatnog objašnjenja što je pošlo po krivu. Kada se naiđe na pogrešku u radu aplikacije, potrebno je raspisati izvješće o pogrešci kako bi se razvojnog programera obavijestilo o pogrešci. Izvješće o pogrešci pronađenoj manualnim i automatiziranim testiranjem prikazano je u Tablici 2., a stvarni rezultat je prikazan na slici 5.2.

Tablica 2. *Izvješće o pogrešci.*

| | |
|---|---|
| **Naslov** | Pronalazak transakcije po ID-u vraća pogrešku |
| **Preglednik** | Google Chrome |
| **Koraci** | 1. Otvori ParaBank aplikaciju<br>2. Prijavi se<br>3. Otvori „Find Transactions" putem navigacije<br>4. U sekciji „Find by Transaction ID" unesi „12345"<br>5. Klikni na „Find Transactions" |
| **Očekivani rezultat** | Prikazana je tablica s rezultatima |
| **Stvarni rezultat** | Prikazana je pogreška |




Slika 5.2. *Pogreška u radu aplikacije.*

Kada se izvješće o pogrešci dostavi razvojnom programeru, njegova je zadaća pronaći rješenje koje će ispraviti pogrešku i omogućiti ispravan rad aplikacije. Po ispravljanju pogreške i *deploy-u* ispravljene aplikacije, zadaća inženjera osiguranja kvalitete jest ponovno testirati

aplikaciju kako bi se potvrdilo da greška u radu više nije prisutna te je stvarno ponašanje aplikacije jednako očekivanom.

Za manualno *smoke* testiranje potrebno je u prosjeku 50 minuta što iznosi prosječno 52 sekunde po testu.

## 5.3. Učinkovitost automatiziranih testova

Iz analize izvođenja automatiziranih i manualnih testova vidljivo je da se isti broj automatiziranih testova proveo u manje vremena nego što je potrebno za izvođenje manualnog testiranja. Ista pogreška u radu pronađena je automatiziranim i manualnim testiranjem, dok se prilikom automatiziranog testiranja pronašlo još dvije pogreške u radu aplikacije koje se nisu pronašle tijekom manualnog testiranja.

Na provedenom testiranju vidljivo je da su automatizirani testovi gotovo osam puta brži u izvođenju, a njima se uspjelo uhvatiti pogrešku koja se pronašla i prilikom manualnog testiranja. Druge dvije pogreške pronađene tijekom automatiziranog testiranja mogu se pripisati nekompatibilnosti aplikacije izvođenju u automatiziranom pregledniku, što bi se svakako prijavilo kao pogreška u radu.

Vrijeme provedeno za razvoj automatiziranih testova u ovom bi se slučaju isplatilo, budući da se *smoke* testiranje provodi često jer su funkcionalnosti međusobno povezane. Automatiziranim *smoke* testiranjem osiguralo bi se da svakom novom promjenom glavne funkcionalnosti rade na očekivani način.

# 6. ZAKLJUČAK

Cilj ovog diplomskog rada je implementacija i provedba automatiziranih testova za *smoke* testiranje internet aplikacije ParaBank. Testiranje internet aplikacija je detaljno analizirano prema vrstama testiranja, pojašnjen je pojam *smoke* testiranja te struktura internet aplikacija. Predstavljene su razlike između manualnog i automatiziranog testiranja te alati za automatizirano testiranje. Opisan je pristup automatiziranom testiranju kao i priprema okruženja za pisanje programskog kôda automatiziranih testova programskim jezikom JavaScript u Cypressu. Objašnjena je struktura *spec* datoteka kao i način dohvaćanja elemenata s internet aplikacije u obliku lokatora. Automatizirani testni slučajevi za *smoke* testiranje raspisani su prema testnim slučajevima za manualno *smoke* testiranje kako bi se automatiziralo testiranje koje za cilj ima osigurati nenarušen rad aplikacije. Rezultati dobiveni automatiziranim *smoke* testiranjem su analizirani te uspoređeni s rezultatima manualnog testiranja.

Ljudski faktor važan je čimbenik u *e2e* testiranju, ali velik broj testova koji ne zahtijevaju stalan ljudski nadzor može se izvesti i evaluirati automatizirano. Veoma je važno odlučiti isplati li se ulagati resurse u razvoj automatiziranih testova ovisno o potrebama aplikacije i trajanju razvoja iste. Analizom učinkovitosti automatiziranih testova na internet aplikaciji ParaBank zaključeno je kako se isplati ulagati u implementaciju automatiziranih *smoke* testova za ovakvu aplikaciju koja ima puno funkcionalnosti te zahtijeva ponavljajuće testiranje glavnih funkcionalnosti. Automatizirani testovi nisu savršeni i zahtijevaju održavanje kao i intervenciju inženjera osiguranja kvalitete ukoliko dođe do pada testova. Nadogradnjom aplikacije potrebno je optimizirati i nadograđivati testove, a kvalitetno pisanje testova neovisnih jednih o drugima pruža veću sigurnost i kvalitetu testiranja. Automatizirani testovi i dalje ne mogu opstati bez intervencije inženjera osiguranja kvalitete niti u potpunosti zamijeniti manualno testiranje. Kombinacija automatiziranih testova s manualnim testovima kojima se testiraju značajke aplikacije za čije je testiranje potrebno ljudsko oko daje optimalne rezultate i pokrivenost svih značajki aplikacije.

# LITERATURA

[1] N. Islam, „A Comparative Study of Automated Software Testing Tools”, Culminating Projects in Computer Science and Information Technology, 2016., dostupno na: https://repository.stcloudstate.edu/csit_etds/12 [24. 8. 2023.]

[2] P. Mahajan, S. Harshal, U. Patkar, „Automation Testing In Software Organization”, International Journal of Computer Applications Technology and Research. str. 198-201, 10.7753/IJCATR0504.1004, 2016., dostupno na: https://www.researchgate.net/publication/299570995_Automation_Testing_In_Software_Organization [24. 8. 2023.]

[3] R. L. Black, E. Van Veenendaal, D. Graham, „Foundations of Software Testing: ISTQB certification”, 4th Edition, str. 1-15, 62-69, 2021.

[4] C. Kaner, J. Bach, B. Pettichord, „Lessons learned in software testing”, str. 93-127, 231-259, 2001.

[5] R. Black, „Managing the testing process: Practical tools and techniques for managing hardware and software testing”, str 1-10, 49-69, 2009.

[6] T. Hamilton, „What is Smoke Testing?”, 2023., dostupno na: https://www.guru99.com/smoke-testing.html [16. 8. 2023.]

[7] A. Petrosyan, „Worldwide digital population 2023”, 2023., dostupno na: https://www.statista.com/statistics/617136/digital-population-worldwide/#:~:text=Worldwide%20digital%20population%202023&text=As%20of%20April%202023%2C%20there,population%2C%20were%20social%20media%20users [6. 8. 2023.]

[8] V. Mannotra, „Guide to Web Application Testing”, 2023., dostupno na: https://www.browserstack.com/guide/web-application-testing [6. 8. 2023.]

[9]. M. Hanna, A. E. Aboutabl , M. M. Mostafa, „Automated Software Testing Framework for Web Applications”, International Journal of Applied Engineering Research ISSN 0973-4562 Volume 13, Number 11, pp. 9758-9767, 2018., dostupno na: https://www.ripublication.com/ijaer18/ijaerv13n11_141.pdf [24. 8. 2023.]

[10] D. S. Nagabushanam, S. Dharinya S, D. Vijayasree, N. S. Roopa, A. Arun, „A Review on the Process of Automated Software Testing” 2022. , dostupno na:

https://www.researchgate.net/publication/363363196_A_Review_on_the_Process_of_Automated_Software_Testing [24. 8. 2023.]

[11] I. Hmelik, „You Don't Need Automated Testing?", 2021., dostupno na: https://www.cobeisfresh.com/blog/you-dont-need-automated-testing [24. 8. 2023.]

[12] P. Akshay, „Manual Testing vs Automation Testing", 2023., dostupno na: https://www.browserstack.com/guide/manual-vs-automated-testing-differences [24. 8. 2023.]

[13] S. Alferidah, S. Ahmed, „Automated Software Testing Tools", 2020. 10.1109/ICCIT-144147971.2020.9213735, dostupno na: https://www.researchgate.net/publication/344311956_Automated_Software_Testing_Tools [24. 8. 2023.]

[14] The Selenium Browser Automation Project, dostupno na: https://www.selenium.dev/documentation [6. 8. 2023.]

[15] Why Cypress?, dostupno na: https://docs.cypress.io/guides/overview/why-cypress [6. 8. 2023.]

[16] Playwright, dostupno na: https://playwright.dev [6. 8. 2023.]

[17] Documentation for Visual Studio Code, dostupno na: https://code.visualstudio.com/docs [16. 8. 2023.]

[18] JavaScript, dostupno na: https://en.wikipedia.org/wiki/JavaScript [16. 8. 2023.]

[19] Getting Started with Faker, dostupno na: https://fakerjs.dev/guide/ [16. 8. 2023.]

[20] ParaSoft Demo Website, dostupno na: https://parabank.parasoft.com/parabank/about.htm;jsessionid=E201C897FD2FD6FF59092CE8FD52616C [27. 8. 2023.]

[21] What is Cypress Page Object Model?, dostupno na: https://www.browserstack.com/guide/cypress-page-object-model [24. 8. 2023.]

[22] How to find an HTML Element using Cypress Locators, dostupno na: https://www.browserstack.com/guide/find-html-element-using-cypress-locators [24. 8. 2023.]

[23] What is End-to-End (E2E) Testing?, dostupno na: https://katalon.com/resources-center/blog/end-to-end-e2e-testing [24. 8. 2023.]

[24] Assertions, dostupno na: https://docs.cypress.io/guides/references/assertions [27. 8. 2023.]

# SAŽETAK

Cilj ovog diplomskog rada jest automatiziranje testiranja za internet aplikaciju ParaBank. Opisana je važnost testiranja i vrste testiranja koje se provode prilikom testiranja aplikacija. Naglasak je na provođenju *smoke* testiranja koje osigurava ispravan rad glavnih funkcionalnosti aplikacije te je analizirano na koji način automatizirano testiranje može rasteretiti inženjera osiguranja kvalitete. Testiranje je provedeno praćenjem testnih slučajeva raspisanih nakon analize aplikacije i njenih funkcionalnosti. Programskim jezikom JavaScript napisan je programski kôd za automatizirane testove korištenjem alata Cypress. Rezultati provođenja automatiziranih testova analizirani su i uspoređeni s rezultatima manualnog testiranja u vidu broja pronađenih pogrešaka u radu aplikacije, trajanju izvođenja testova i broja pokrivenih testnih slučajeva. Analizom rezultata donesen je zaključak o isplativosti provođenja automatiziranog testiranja.

**Ključne riječi:** analiza, Cypress, internet aplikacija, kvaliteta, testiranje

# ABSTRACT

**Title: Automated smoke testing of Internet applications**

The goal of this thesis is the automation of testing for the Internet application ParaBank. The importance of testing and the types of testing that are performed during application testing are described. The emphasis is on the implementation of smoke testing, which ensures the correct behavior of the main functionalities of the application, and it is analyzed how automated testing can relieve the Quality Assurance Engineer. The testing was conducted by following the test cases written after analysis of application and its functionalities. Programming code for automated tests using Cypress tool was written in the JavaScript programming language. The results of conducting automated tests were analyzed and compared with results of manual testing in the form of the number of errors found in application, the duration of the tests and the number of covered test cases. By analyzing the results, a conclusion was reached about profitability of conducting automated testing.



**Keywords**: analysis, Cypress, Internet application, quality, testing

# ŽIVOTOPIS

Amela Vorgić rođena je 27. rujna 1999. godine u Osijeku. Pohađa Osnovnu školu Drenje te 2014. godine upisuje jezični program Gimnazije Antuna Gustava Matoša u Đakovu. Po završetku srednje škole ostvaruje pravo na izravan upis na Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek gdje 2018. godine upisuje sveučilišni preddiplomski studij Računarstva koji završava 2021. godine. Iste godine upisuje sveučilišni diplomski studij Računarstva, izborni blok Programsko inženjerstvo.

# PRILOZI

Prilog 1. Diplomski rad

Prilog 2. Programski kôd

https://github.com/AmelaVorgic/SmokeTesting-ParaBank