CYPRESS-JAVASCRIPT PROGRAMSKI OKVIR ZA "END TO END" TESTOVE

Žitko, Nikola

Undergraduate thesis / Završni rad

2022

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Split / Sveučilište u Splitu**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:228:273377

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-05-06**

Repository / Repozitorij:

Repository of University Department of Professional Studies

zir.nsk.hr

UNIVERSITY OF SPLIT

dabar
DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Preddiplomski stručni studij Informacijske tehnologije

NIKOLA ŽITKO

Z A V R Š N I R A D

CYPRESS – JAVASCRIPT PROGRAMSKI OKVIR ZA „END TO END“ TESTOVE

Split, srpanj 2022.

SVEUČILIŠTE U SPLITU
SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Preddiplomski stručni studij Informacijske tehnologije

Predmet: Operativni sustavi

Z A V R Š N I R A D

Kandidat: Nikola Žitko

Naslov rada: Cypress – JavaScript programski okvir za „End to End“ testove

Mentor: Ljiljana Despalatović, v. pred.

Split, srpanj 2022.

Sadržaj

Sažetak **4**
Summary **5**
1. Uvod **6**
2. Pristup testiranju **7**
2.1. „White-box“ testiranje 7
2.2. „Black-box“ testiranje 8
3. Razine testiranja **9**
3.1. Jedinično testiranje 9
3.2. Integracijsko testiranje 9
3.3. Testiranje sustava 10
4. Vrste testiranja **11**
4.1. „Smoke“ testiranje 11
4.2. Regresijsko testiranje 11
4.3. Automatizirano testiranje 12
4.4. „End to End“ testiranje 12
5. JavaScript **14**
5.1. Povijest 14
5.2. Princip rada 14
5.3. Karakteristike JavaScripta 15
6. Cypress **17**
6.1. Povijest Cypressa 17
6.2. Karakteristike Cypressa 18
6.3. Značajke Cypressa 19
6.4. Vrste testova 19
6.5. Postavljanje Cypressa 21
6.6. End To End Testovi 25
6.6.1. Pisanje testova 26
6.6.2. Pokretanje testova 32
6.6.3. Analiziranje testova 34
6.7. Primjer korištenja 36
7. Zaključak **37**
Literatura **38**

Sažetak

U ovome radu predstavljen je JavaScript razvojni okvir za testiranje Cypress. U radu su opisani pristupi testiranju, razine i vrste testiranja, programski jezik JavaScript, te razvojni okvir za testiranje Cypress. U radu je prikazano kako Cypress funkcionira, koje su sve njegove karakteristike i značajke, prednosti i mane, te kako se testovi pišu, pokreću i zatim analiziraju. Na kraju će biti prikazan primjer korištenja Cypressa u stvarnom okruženju. Korištene tehnologije su programski jezik JavaScript za pisanje kôda, te razvojni okvir Cypress za pisanje, pokretanje i analiziranje testova.

Ključne riječi: *JavaScript, Cypress, testiranje, aplikacija*

Summary

Cypress – JavaScript „End to End“ testing framework

This paper presents Cypress, a JavaScript testing framework. This paper describes testing approaches, testing levels and types, JavaScript framework, and Cypress testing framework. This paper describes the inner workings of Cypress and its features, as well as its pros and cons. Paper also describes how tests are written, run and debugged. Lastly, a real world example of using Cypress is shown. Technologies used are programming language Javascript for writing code and Cypress framework for writing, running and debugging tests.

Keywords: *JavaScript, Cypress, testing, application*

1. Uvod

U današnjem svijetu postoje desetci različitih načina razvoja aplikacija. Velikoj većini je zajedničko da je jedan od koraka osiguranje kvalitete (engl. *Quality Assurance*). Osiguranje kvalitete je usko povezano sa svim drugim koracima razvoja aplikacije. Od planiranja i procjene razvoja aplikacije, dizajna i korisničkog iskustva, sve do testiranja same aplikacije i njezinih funkcionalnosti.

Kako bi se brzo i efikasno razvijala aplikacija, bitno je da prilikom razvoja bude što manje pogrešaka u kôdu aplikacije kako bi ona funkcionirala ispravno. Ukoliko aplikacija radi kako treba u svakom trenutku, ona ne zahtjeva popravke i preinake koji oduzima veliku količinu skupocjenog vremena. Ispravnost aplikacija se osigurava raznim načinima testiranja koje ćemo detaljno opisati u ovom radu. Jedan od načina su automatizirani testovi.

Automatizirani testovi omogućuju korisniku da brzo, efikasno i točno testira aplikaciju u potpunosti. Upravo ta radnja omogućava ogromnu uštedu na skupocjenom vremenu u modernom razvoju aplikacija.

Rad je podijeljen u pet cjelina. Drugo poglavlje opisuje različite pristupe testiranju, te njihove prednost i mane. Treće i četvrto poglavlje opisuje razine i vrste testiranja, od ručnog testiranja samo pojedine funkcionalnosti do potpuno automatiziranog procesa testiranja. Peto poglavlje opisuje programski jezik JavaScript koji je temelj razvojnog okvira za testiranje Cypress. Šesto poglavlje detaljno opisuje razvojni okvir za testiranje Cypress, njegovi funkcionalnosti, značajke i karakteristike. Detaljno opisuje i „End to End“ testiranje, te pisanje, pokretanje i analiziranje „End to End“ testova preko praktičnog primjera. U završnom poglavlju je donesen zaključak rada.

2. Pristup testiranju

Dostupno je mnogo različitih pristupa softverskog testiranja. Možemo ih podijeliti na statičke i na dinamičke pristupe. Statički pristupi uključuju čitanje i provjeru samog kôda, bilo to provjera ručno čitajući kôd ili korištenjem raznih programskih alata ili programa za obradu teksta. Dinamički pristupi uključuju testiranje pokrenutog programa ili aplikacije. Dinamičko testiranje može započeti i prije nego je program stopostotno gotov ukoliko se testiraju određene funkcionalnosti u zasebnom okruženju.

Jedan od najpopularnijih pristupa je pristup bijele i crne „kutije“ (engl. *White-box, Black-box*) [1]. Ovaj pristup opisuje pogled osobe koja piše testova kada dizajnira testne slučajeve.

2.1. „White-box“ testiranje

„White-box“ testiranje, poznato kao i prozirno ili strukturalno testiranje, testira i potvrđuje internu strukturu i funkcioniranje programa. Kod „white-box“ testiranja prilikom dizajniranja testnih slučajeva uzimaju se u obzir izvorni kôd i same vještine programiranja. Osoba koja piše testove odabire ulazne podatke te određuje izlazne podatke ovisno o putanji kroz kôd programa.

„White-box“ testiranje se primarno koristi kod pisanja jediničnih testova (testova najniže razine) koji testiraju najosnovnije funkcionalnosti programa. Iako se izvode brzo i mogu uhvatiti mnogo problema i grešaka, ne mogu detektirati greške kod složenijih programa gdje različite komponente međusobno komuniciraju.

U današnjem svijetu osim jediničnih testova, pristup bijele „kutije“ se koristi za testiranje API (engl. *Application Programming Interface*) sučelja, testiranje uspješnog izvršavanja svih dijelova programa i testiranje namjerno izazvanih pogreški. Upravo ovo testiranje izvršavanja svih dijelova programa je izrazito korisno u pronalaženju nepotrebnog kôda programa koji se može izbaciti ili optimizirati.

2.2. „Black-box“ testiranje

„Black-box“ testiranje, poznato kao i funkcionalno testiranje, razmatra program kao „crnu kutiju“, ispitivajući funkcionalnost programa bez ikakvog znanja o unutarnjoj implementaciji ili izvornom kôdu. Osobe koji dizajniraju testne slučaje upoznati su jedino s funkcionalnošću programa, odnosno što program radi, ali ne i kako to radi.

Jednostavno se provjerava da li određena ulazna vrijednost ili informacija rezultira s ispravnom izlaznom vrijednosti ili informacijom. Ovo pojednostavljuje testiranje jer je rezultat isključivo ili ispravan ili neispravan.

Jedna od prednosti „black-box“ testiranja je što ne zahtjeva znanje o programiranju. Neovisno kakve sklonosti imaju programeri, tester će imati drukčije i naglasit će testiranje drugih područja funkcionalnosti koje su programeri možda predvidjeli.

S druge strane, „black-box“ testiranje se često opisuje kao „kretanje mračnim labirintom bez svjetiljke“. Kako testeri ne pregledavaju izvorni kôd, moguće je da će kreirati višak testnih slučajeva koji testiraju istu funkcionalnost ili će skroz propustiti neku funkcionalnost.

3. Razine testiranja

Generalno govoreći postoje tri razine testiranja: jedinično testiranje, integracijsko testiranje i testiranje sustava. Doduše, četvrtu razinu, ispitivanje prihvatljivosti, programeri znaju također uključiti u razine testiranja [2]. Testovi su uobičajeno grupirani po ovim razinama, ovisno kada su dodani u razvojnom procesu ili ovisno o specifičnoj razini samog testa.

3.1. Jedinično testiranje

Prethodno spomenuti, jedinični testovi se odnose na testove koji provjeravaju funkcionalnost specifičnog dijela kôda, najčešće na funkcionalnoj razini. U objektno-orijentiranom okruženju, to je najčešće na razini klasa, a minimalni test uključuje konstruktore i destruktore.

Jedinični testovi su najčešće pisani od strane samih programera dok rade na samom kôdu („white-box“ pristup), kako bi osigurali da određena funkcionalnost ispravno funkcionira. Jedinični testovi ne mogu sami po sebi provjeriti cjelokupni program, ali mogu provjeriti temeljne blokove.

Jedinično testiranje je proces razvoja softvera koji uključuje široki spektar strategija za detektiranje i preventiranje pogrešaka kako bi se smanjio razvojni rizik, razvojno vrijeme i cijena. Cilj jediničnog testiranja je da eliminira temeljne greške prije nego se provuku do idućih razina testiranja. Namjena strategije je da poveća kvalitetu krajnjeg softvera kao i efikasnost cjelokupnog razvojnog procesa.

3.2. Integracijsko testiranje

Integracijsko testiranje je bilo koja vrsta softverskog testiranja kojoj je cilj potvrditi ispravnost sučelja između komponenti programa. Cilj integracijskih testova je da otkriju pogreške u sučeljima i komunikaciji između različitih sučelja, komponenti ili modula.

Progresivno sve veće grupe testiranih komponenti programa se integriraju i ponovno testiraju sve dok cijeli softver ne funkcionira kao sustav.

3.3. Testiranje sustava

Testiranje sustava je vrlo jednostavno. Ono uključuje sveobuhvatno testiranje kompletnog gotovog sustava kao cjelinu kako bi se provjerilo da taj sustav zadovoljava sve prethodno definirane zahtjeve koje taj sustav treba obavljati.

4. Vrste testiranja

Postoji desetak različitih vrsta testiranja. Neke se stalno koriste, a neke služe za samo vrlo specifične situacije. U idućem dijelu teksta ćemo detaljno proći kroz nekoliko najbitnijih. To su „Smoke“ testiranje, regresijsko testiranje, automatizirano testiranje i „End to End“ testiranje.

4.1. „Smoke“ testiranje

„Smoke“ testiranje, poznato kao i testiranje povjerenja ili test provjere izrade, je preliminarno testiranje kojem je cilj otkriti jednostavne greške koje su dovoljno ozbiljne da odbace izdavanje nove verzije softvera. „Smoke“ testiranje je podgrupa testnih slučajeva koji pokrivaju najvažnije funkcionalnosti programa.

Cilj „Smoke“ testiranja je da utvrdi je li uopće potrebno detaljnije testirati program. Na primjer, ukoliko se program uopće ne pokreće ili se korisnik ne može prijaviti, nema svrhe ići na sljedeću detaljniju razinu testiranja. Na taj način se brzo pregledava površno stanje programa, te može uštedjeti dragocjeno vrijeme.

„Smoke“ testovi se često pokreću, a kako kratko traju, brzo daju korisnu povratnu informaciju. Microsoft je izjavio kako nakon pregledavanja kôda, „smoke“ testiranje je najisplativija metoda za identificiranje i ispravljanje greški u programu [3].

4.2. Regresijsko testiranje

Regresijsko testiranje je ponovno testiranje prethodno razvijenog i testiranog softvera kako bi se ustanovilo radi li sve ispravno nakon napravljenih promjena. Ukoliko ne radi, imamo regresiju.

Razlozi koji mogu dovest do potrebe regresijskog testiranja su popravci grešaka, nadogradnje softvera, promjene konfiguracija, čak i zamjena elektroničkih komponenti.

Svaka promjena može uzrokovati da neki drugi, izgledom nepovezani, dio programa prestane raditi ispravno. Kako bi se to izbjeglo, regularno se obavljaju ili pokreću regresijski testovi. Najčešće svaki dan, svaki tjedan ili prilikom nadogradnje softvera.

Regresijsko testiranje se može obavljati ručno ali može biti i automatizirano pomoću alata za automatizaciju [3].

4.3. Automatizirano testiranje

Automatizirano testiranje uključuje korištenje zasebnog softvera koji kontrolira pokretanje testova i usporedbu rezultata testova s očekivanim rezultatima. Automatizirano testiranje, kao što mu i sam naziv govori, može automatizirati repetitivne ali nužne testove ili izvršiti suviše kompleksne testove koji su previše zahtjevni za ručno testiranje. Automatizirano testiranje je kritično za kontinuirano razvijanje i puštanje novog softvera korisnicima.

Mnogo je različitih pristupa automatizaciji testova, ali dva pristupa se većinski koriste. To su testiranje preko korisničkog sučelja (engl. *Graphical User Interface testing*) i testiranje preko API sučelja.

Testiranje preko korisničkog sučelja uključuje generiranje događaja, poput klika mišem ili pritiska na tipku na tipkovnici, nakon kojeg se promatra promjena u sučelju, kako bi se utvrdila ispravnost programa.

Testiranje preko API sučelja uključuje testiranje pomoću API poziva koji kompletno zaobilaze korisničko vizualno sučelje. Povratni odgovori s API sučelja se uspoređuju s očekivanim te se utvrđuje ispravnost programa [3].

4.4. „End to End“ testiranje

„End to End“ testiranje (u nastavku teksta E2E), poznato i kao „End-to-End“ ili „E2E“ testiranje, se odnosi na metodu testiranja softvera koji uključuje testiranje cijelog aplikacijskog tijeka rada, od početka do kraja. Ova metoda pokušava replicirati scenarije

pravog korisnika, kako bi se ispravnost sustava u potpunosti provjerila. U načelu, test prolazi kroz sve operacije aplikacije kako bi se provjerili svi dijelovi aplikacije, od hardvera, baze podataka, komunikacije između sučelja, do mrežne povezanosti i vanjskih ovisnosti prema drugim aplikacijama [4].

E2E testovi vremenski traju najduže od svih testova jer repliciraju pravog korisnika i izvršavaju se u stvarnom vremenu. To uključuje čekanje podataka preko internetske veze ili fizičkog podatkovnog diska i njihovo učitavanje. Zbog toga se E2E testovi rjeđe pokreću, svako par tjedana ili prilikom većeg ažuriranja softvera, ali zato daju ogroman benefit jer ukoliko su ispravno napisani da pokriju cijelu aplikaciju, biti ćemo sigurni da ona u potpunosti funkcionira ispravno.

S E2E testovima se teži da u potpunosti zamjene ručno testiranje. Iako zahtijevaju određenu, nekad i veliku količinu vremena i resursa, vrlo brzo se isplate, iako na prvu ne izgleda tako. Na primjer, zašto potrošiti deset sati na pisanje jednog testnog slučaja kada nam treba petnaest minuta da ručno testiramo taj testni slučaj. Ukoliko izračunamo, vidjet ćemo da nakon samo četrdeset pokretanja tog E2E testa, već će nam vratiti uloženo vrijeme, a nije od nas zahtijevalo da ručno testiramo i utrošimo po petnaest minuta za svako testiranje koje smo mogli utrošiti na nešto drugo.

Kako bi mogli pisati E2E testove, potreban nam je razvojni okvir za pisanje automatiziranih testova. Jedan od njih je razvojni okvir za testiranje Cypress o kojem ćemo detaljno pisati u nastavku ovog rada. Prije nego to možemo, trebamo se upoznati s JavaScript programskim jezikom kojeg koristi razvojni okvir za testiranje Cypress.

5. JavaScript

Programski jezik JavaScript, poznat kao i JS, je jedan od temeljnih tehnologija interneta i svjetske mreže (engl. World Wide Web). 2022. godine, devedeset i osam posto (98%) internetskih stranica koristi JavaScript programski jezik visoke razine koji odgovara ECMAScript standardu. Sadrži dinamičku provjeru napisanog kôda (provjera točnosti u stvarnom vremenu), prototipno baziranu objektnu orijentaciju i funkcije prve klase (funkcije mogu biti argumenti drugih funkcija).

5.1. Povijest

Tijekom ranih devedesetih godina, na svjetskoj mreži jedino su postajale statične internet stranice. Nisu imale mogućnost nikakvog dinamičkog ponašanja nakon što bi se stranica učita u pregledniku. Kako bi se to promijenilo i kako bi makli to ograničenje, korporacija Netscape, 1995. godine dodaje skriptni jezik u tada popularni preglednik Navigator. U tom trenutku nastaje JavaScript skriptni jezik, koji se tijekom godina razvija u potpuni programski jezik [5].

5.2. Princip rada

JavaScript prilikom izvođenja se sastoji od dva dijela. Prvi dio je memorijski spremnik, a drugi dio je spremnik izvršavanja kôda. Internet preglednik kreira ova dva spremnika. U prvom koraku, JavaScript pregleda i učita sve varijable i funkcije u memorijski spremnik. Zatim se slijedno izvršava napisati kôd koji se učitava u spremnik izvršavanja kôda. JavaScript je jednonitni jezik. Odnosno, samo jedna naredba se može istovremeno izvršavati [6].

5.3. Karakteristike JavaScripta

HTML (engl. *HyperText Markup Language*) nema mogućnosti da izvršava logičke operacije, poput uvjetne provjere, programske petlje, uvjetno odlučivanje, te matematičke operacije (zbrajanje, oduzimanje, množenje, dijeljenje, itd.). Tu nam uskače JavaScript.

JavaScript je jednonitni asinkroni programski jezik. Jednonitni označava da se napisani kôd izvršava slijedno, liniju po liniju. Sljedeća linija kôda se ne može izvršiti prije prethodne. Iako je JS jednonitni programski jezik, on je i asinkroni uz pomoć tzv. petlje događaja. Kada spremnik izvršavanja kôda prepozna asinkronu funkciju, on će je proslijediti internetskom pregledniku te će je dodati u petlju događaja. Asinkrona funkcija će se tada izvršavati u pozadini, te će JavaScript nastaviti izvršavati dalje kôd. Kada se izvrši asinkrona funkcija, petlja događaja će vratiti rezultat izvršavanja natrag JavaScriptu (Ispis 1.).

JavaScript nam omogućava da kreiramo interaktivne *web* stranice. Omogućava kreiranje i prikazivanje raznih dijaloških okvira, otvara i zatvaranje prozora preglednika, animacije sadržaja stranice, dinamičko prikazivanje sadržaja, validaciju unosa, itd.

Primjer zbrajanja dva broja je dan u ispisu 2, a primjer prikazivanja trenutnog datuma i vremena je dan u ispisu 3.

```
console.log("prvi")

setTimeout(() => {
    console.log("drugi")
}, 1000)

console.log("treći")

//rezultat izvođenja

prvi
treći
drugi
```

Ispis 1: Primjer asinkronog izvršavanja JavaScript kôda

```
const broj1 = 5;
const broj2 = 3;

// zbroji dva broja
const suma = broj1 + broj2;

// prikazi zbroj
console.log('Suma brojeva ' + broj1 + ' i ' + broj2 + ' je: ' + suma)

// rezultat
Suma brojeva 5 i 3 je: 8
```

Ispis 2: Primjer zbrajanja dva broja u programskom jeziku JavaScript

```
// program koji ispisuje trenutni datum i vrijeme
// dohvatimo trenutni datum i vrijeme
const date = new Date();

// spremimo datum u zasebnu varijablu
const datum = date.toDateString();

// spremimo vrijeme u zasebnu varijablu
const vrijeme = date.toLocaleTimeString();

// prikazemo datum
console.log('Datum: ' + datum);

// prikazemo vrijeme
console.log('Vrijeme: ' + vrijeme);
```

Ispis 3: Primjer prikazivanja trenutnog datuma i vremena

6. Cypress

Razvojni okvir za testiranje Cypress je alat nove generacije za testiranje napravljen za moderni *web*. Cypress omogućava postavljanje i pisanje testova, pokretanje testova i analiziranje svakog koraka završenog testa [7].

Cypress većinski koriste programeri ili inženjeri osiguranja kvalitete koji razvijaju internet aplikacije koristeći razvojne okvire JavaScripta. Cypress omogućava pisanje „End to End“ testova, integracijskih testova i jediničnih testova. U ovome radu ćemo se fokusirati na „End to End“ testove.

Cilj Cypressa je da stvori eko sistem otvorenog kôda koji će unaprijediti produktivnost, učiniti testiranje ugodnim iskustvom i na posljetku usrećiti programera. Jedan od koraka do tog cilja je izvrsna i pristupačna dokumentacija. Dokumentacija je napravljena na način da ne opisuje samo kako nešto funkcionira, već opisuje i razloge zašto se određene stvari rade na određeni način.

6.1. Povijest Cypressa

Cypress.io, tvrtka iza razvojnog okvira za testiranje Cypress, je osnovana 2015. godine s ciljem da napravi novi, bolji i efikasniji, način za testiranje. Cypress od 2017. godine lagano dobiva na popularnosti, paralelno sa razvojem novih modernih razvojnih okvira JavaScripta, poput *React* ili *Vue.JS*. Popularnost mu raste i dalje, te 2022. godine ima ogromnih četiri milijuna preuzimanja svaki tjedan. Trenutno broje dvadeset i šest zaposlenih koji regularno ažuriraju Cypress s novim mogućnostima. Cypress reklamiraju kao „Alat za testiranje napravljen za ljude.“ (engl. „*A test runner built for humans.*“) što upravo točno opisuje njihov cilj s ovim alatom [7]. To je visoka pristupačnost, jednostavnost pisanja i pokretanja testova, te analiza izvršenih testova s pregršt informacija.

6.2. Karakteristike Cypressa

Većina alata za „End to End“ testove su bazirani na *Selenium* alatu. *Selenium* je stari alat koji ima određene probleme, a kako je većina bazirana na njemu, svi dijele te iste probleme. Cypress je potpuno novi alat, koji nije baziran na *Selenium* alatu. Kreirana je potpuno nova arhitektura od temelja. Za razliku od *Selenium* alata koji pokreće naredbe preko mreže, Cypress pokreće identične naredbe kao i sami korisnik.

Cypressu nije cilj da bude najbolji generalni alat za testiranje. Nije mu cilj podržavati jedinične testove za poslužitelje ili druge specifične situacija. Cilj mu je biti najbolji alat za „End to End“ testove. Kako je fokusiran na samo jednu specifičnu granu testiranja, to mu omogućava da bude i najbolji.

Cypress može testirati sve što se pokreće u internet pregledniku. Cijela arhitektura Cypressa je napravljena da podržava moderne razvojne okvire JavaScripta. Cypress vrhunski funkcionira sa *React, Angular, Vue, Elm,* itd. razvojnim okvirima. Cypress također funkcionira i sa starijim poslužiteljskim stranicama i aplikacijama.

Iako je moguće prevoditi druge programske jezike u JavaScript, u načelu se JavaScript izvodi u internet pregledniku. Zbog toga se testovi u Cypressu isključivo pišu u JavaScript programskom jeziku.

Pisanje „End to End“ testova inače zahtjeva kombinirani rad mnogo različitih alata. Cypress sam po sebi sadrži sve potrebne alata i ne zahtjeva postavljanje desetke drugih alata ili biblioteka. Najbolji alati i biblioteke su već ugrađeni u Cypress i napravljeni su da međusobno fluidno funkcioniraju.

Jedan od ciljeva Cypressa je stvoriti razvojno okruženje kojeg pokreće testiranje. Daje snagu programerima i inženjerima osiguranja kvalitete da razvijaju aplikacije i programe najbrže što je moguće.

Zbog potpuno nove arhitekture, Cypress testovi se izvode puno brže od klasičnih testova. Cypress omogućava istovremeno testiranje i razvijanje. To je moguće jer je aplikacija vidljiva programeru tijekom izvršavanja testova. Programer i tijekom izvršavanja testova ima pristup svih razvojnim alatima, a promjene koje napravi su odmah vidljive. Rezultat je brži i bolji razvoj aplikacija [7].

6.3. Značajke Cypressa

Sljedeće značajke su navedene na internetskoj stranici Cypressa [7].

- **Putovanje kroz vrijeme:** Cypress sprema interaktivni snimak svakog koraka testa, što omogućava naknadno detaljno pregledavanje što se izvodilo u svakom koraku testa.
- **Analiza testa:** Cypress detaljno prikazuje greške prilikom izvođenja testa. Uključuje i opis greške, vrijeme i lokaciju u kôdu, te čak i uzrok pogreške, kako bi smanjio vrijeme potrebno za pronalaženje problema u kôdu.
- **Automatsko čekanje:** U Cypress testovima nije potrebno ručno dodavati naredbe za čekanje izvršavanja kôda ili učitavanje stranice. Cypress automatski čeka da se sav potreban sadržaj učita prije nego krene utvrđivati ispravnost.
- **Špijuni i satovi:** Mogućnost provjere ponašanja funkcija, odgovora poslužitelja i tajmera.
- **Kontrola mrežnog prometa:** Cypress nudi mogućnost potpune kontrole nad mrežnim prometom aplikacije.
- **Konzistentni rezultati:** Arhitektura ne koristi *Selenium* što znači da su testovi brzi, konzistentni i pouzdani.
- **Snimke i video zapisi:** Cypress prilikom neuspješno izvršenog testa automatski radi snimku i video zapis pogreške, kako bi se lakše mogla analizirati pogreška.
- **Testiranja na više preglednika:** Cypress podržava pokretanja unutar Firefox, Chrome, Edge i svih Electron preglednika.

6.4. Vrste testova

Cypress se može koristiti za pisanje raznih različitih vrsta testova. To dodatno povećava povjerenje da će aplikacija koja se testira raditi ispravno i pod opterećenjem.

Cypress je originalno dizajniran za pokretanje „End to End“ testova za sve što se može pokrenuti u Internet pregledniku. Tipični E2E test posjećuje aplikaciju u pregledniku

i izvodi radnje preko korisničkog sučelja kao što bi i pravi stvarni korisnik. Primjer „End to End“ testa je dan u ispisu 4.

```
it('adds todos', () => {
  cy.visit('https://todo.app.com')
  cy.get('[data-testid="new-todo"]')
    .type('write code{enter}')
    .type('write tests{enter}')
  // potvrdimo da aplikacija prikazuje dvi stavke
  cy.get('[data-testid="todos"]').should('have.length', 2)
})
```

Ispis 4: Primjer „End to End“ testa

Uz „End to End“ testove, Cypress može testirati i komponente podržanih razvojnih okvira. Primjer testa je dan u ispisu 5.

```
import TodoList from './components/TodoList'

it('contains the correct number of todos', () => {
  const todos = [
    { text: 'Buy milk', id: 1 },
    { text: 'Learn Component Testing', id: 2 },
  ]

  cy.mount(<TodoList todos={todos} />)
  // komponenta se pokreće kao aplikacija
  cy.get('[data-testid="todos"]').should('have.length', todos.length)
})
```

Ispis 5: Primjer testa komponente

Također, moguće je pisati i API testove. Cypress može slati i primati HTTP pozivi, što mu dozvoljava da testira i API sučelja. Primjer API testa je dan u ispisu 6.

```
it('adds a todo', () => {
  cy.request({
    url: '/todos',
    method: 'POST',
    body: {
      title: 'Write REST API',
    },
  })
    .its('body')
    .should('deep.contain', {
      title: 'Write REST API',
      completed: false,
    })
```

Ispis 6: Primjer API testa

Uz ove tri vrste testova, Cypress dozvoljava dodavanje i drugih preko dodataka treće strane. Kako je Cypress otvorenog kôda, svatko može napisati svoju vrstu testa. Jedni od najpopularnijih dodataka su vizualni testovi, testovi elektroničke pošte i testovi pristupačnosti.

6.5. Postavljanje Cypressa

Cypress je moguće postaviti na dva načina, preko „*npm"* (engl. *node package manager*) paketa ili preko direktnog preuzimanja.

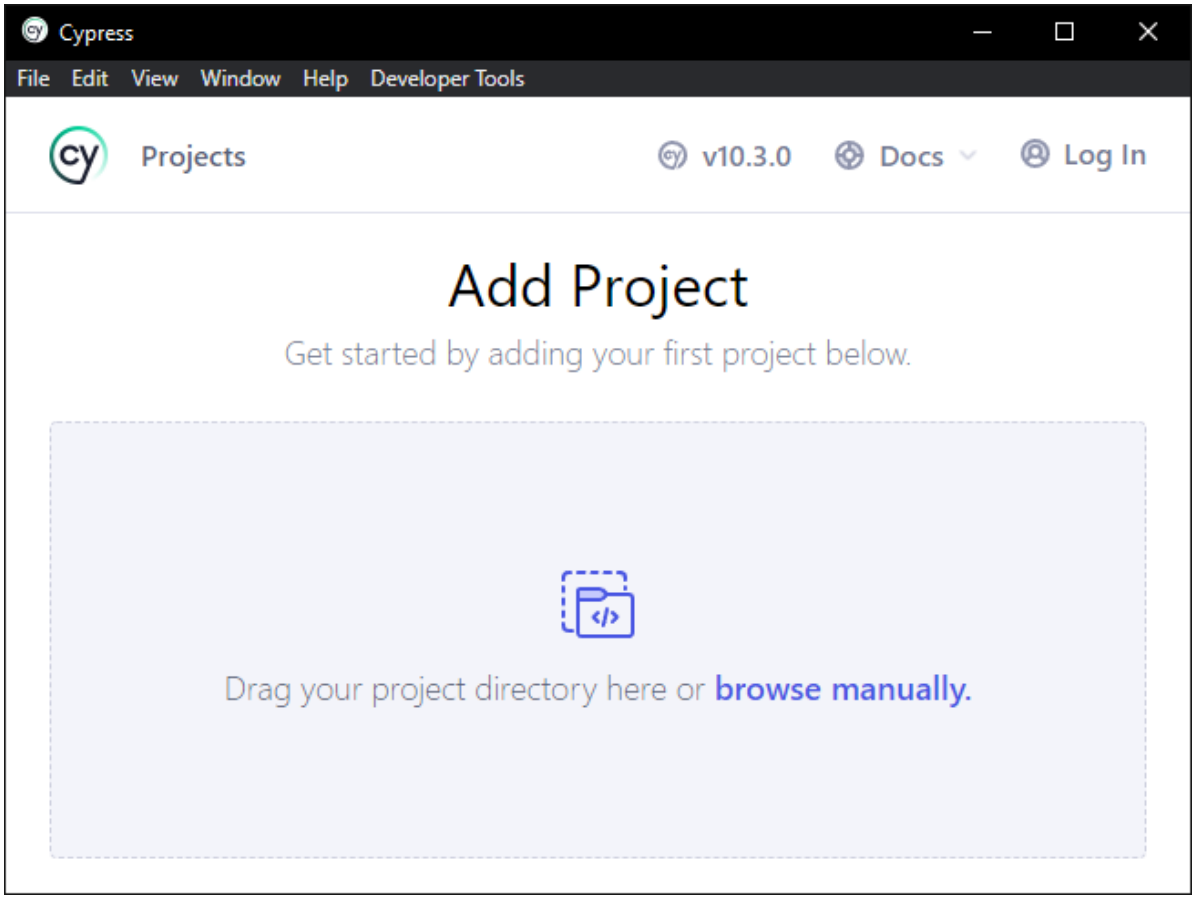
Uobičajeno je dodati Cypress u već postojeći JavaScript projekt koji ima postavljeni Node.JS. U tom slučaju, postavljanje Cypressa kao „*npm*" paketa, otključava sve njegove mogućnosti. To je upravo prikazano u ispisu 7.

```
cd /your/project/path
npm install cypress --save-dev
```

Ispis 7: Postavljanje Cypressa preko „npm" paketa

Direktno preuzimanje služi više kao brzo testiranje i isprobavanje samog Cypressa za korisnike koji se žele upoznati sa njim prije nego ga dodaju kao dio svog projekta. U ovome radu ćemo upravo to napraviti, direktno preuzeti i pokrenuti jer nećemo raditi na osobnom projektu, već ćemo posjećivati već postojeće *web* stranice i njih testirati.

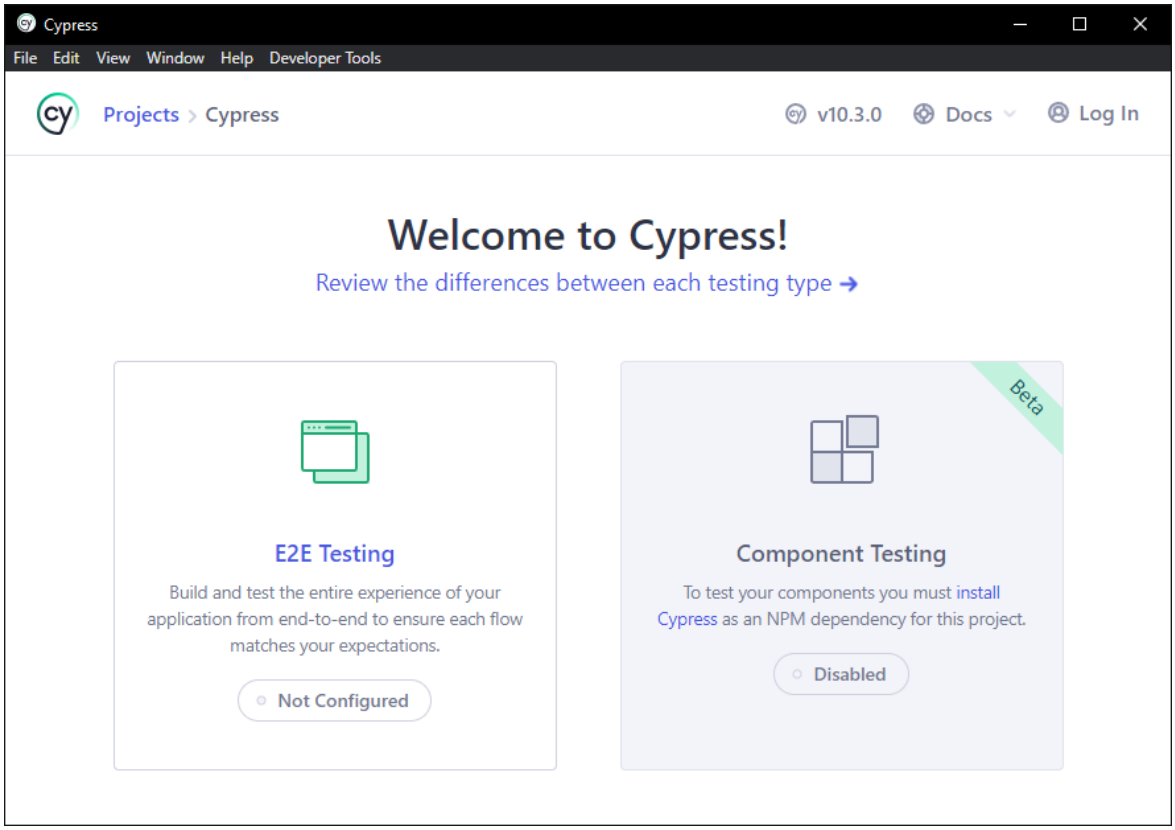
Kako bi preuzeli Cypress, sve što trebamo je posjetiti poveznicu `https://download.cypress.io/desktop`. Stranica će automatski preuzeti ispravnu verziju Cypressa za naš operativni sustav. Preuzeti datoteku ćemo otpakirati i zatim pokrenuti „*Cypress.exe*“ u slučaju *Windows* operativnog sustava. Trebao bi nam se otvoriti prozor na slici 1.



Slika 1: Početni prozor Cypressa

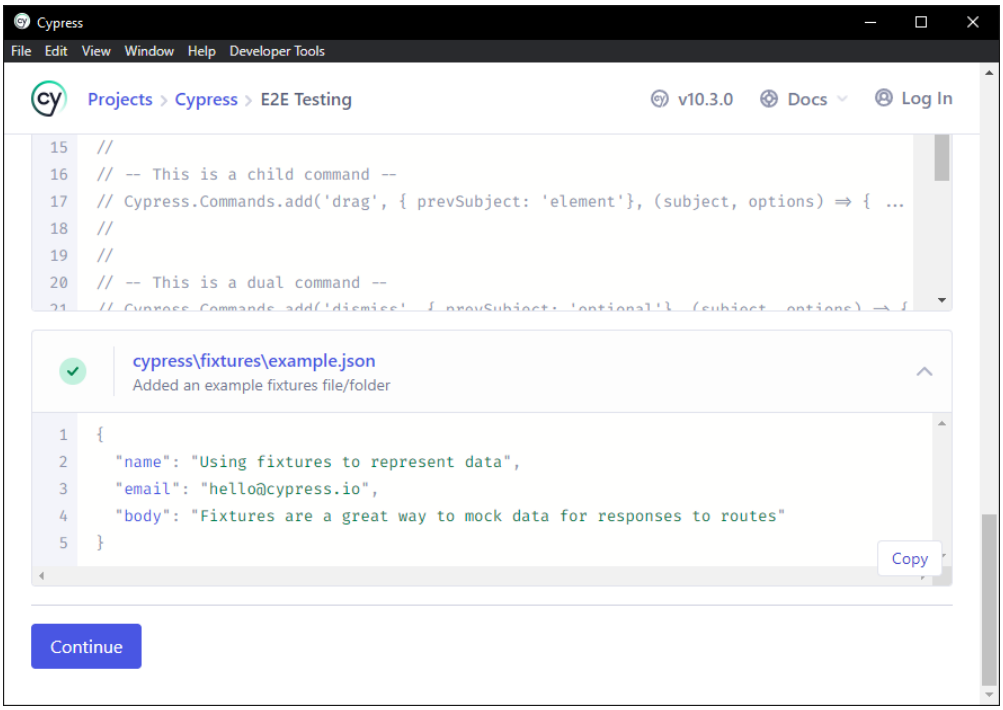
Sada kada smo otvorili Cypress, možemo kreirati projekt i započeti sa pisanjem testova. Kako bi to učinili, kliknut ćemo na botun „browse manually“.

Kada nam se otvori novi dijaloški okvir, kreirati ćemo novu mapu u kojoj ćemo spremiti projekt. Nakon što učinimo, otvoriti će nam se prozor kao na slici 2.



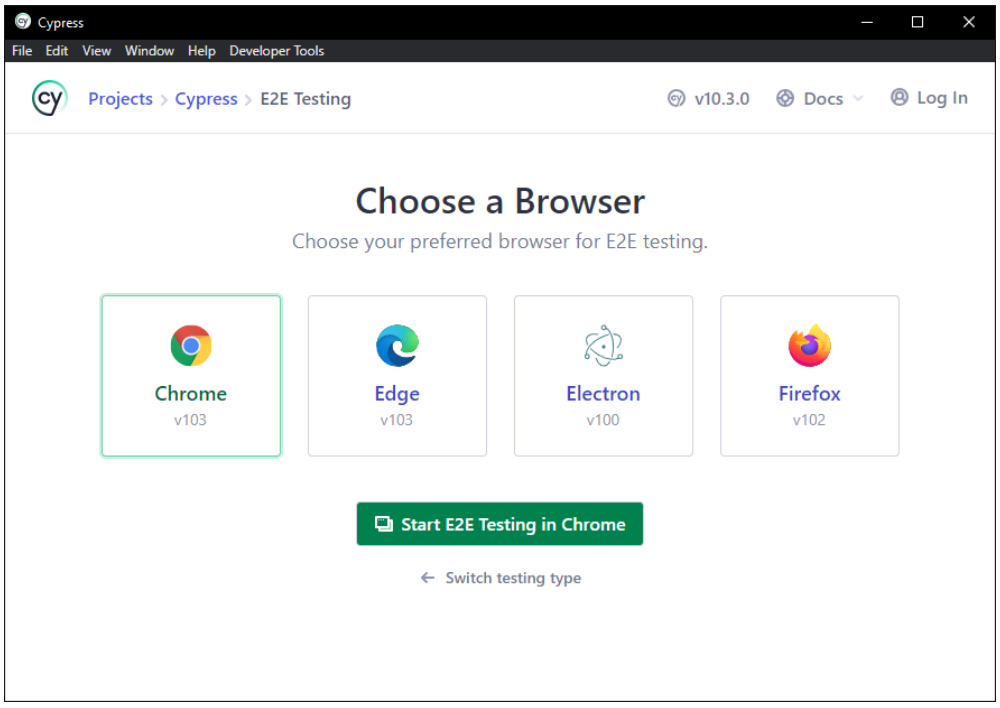
Slika 2: Cypress prozor nakon kreiranja projekta

U ovom trenutku imamo novi prazni projekt. Kako bi mogli započeti s pisanjem testova, potrebno je postaviti različite konfiguracijske datoteke. Na našu sreću, Cypress to obavlja automatski. Sve što trebamo je kliknuti na „E2E Testing“, što ćemo i učiniti. U prozoru će nam se prikazati sve datoteke koje je Cypress automatski kreirao. Sve što još trebamo je na dnu prozora kliknuti na „Continue“ botun, kako bi nastavili dalje (Slika 3.).



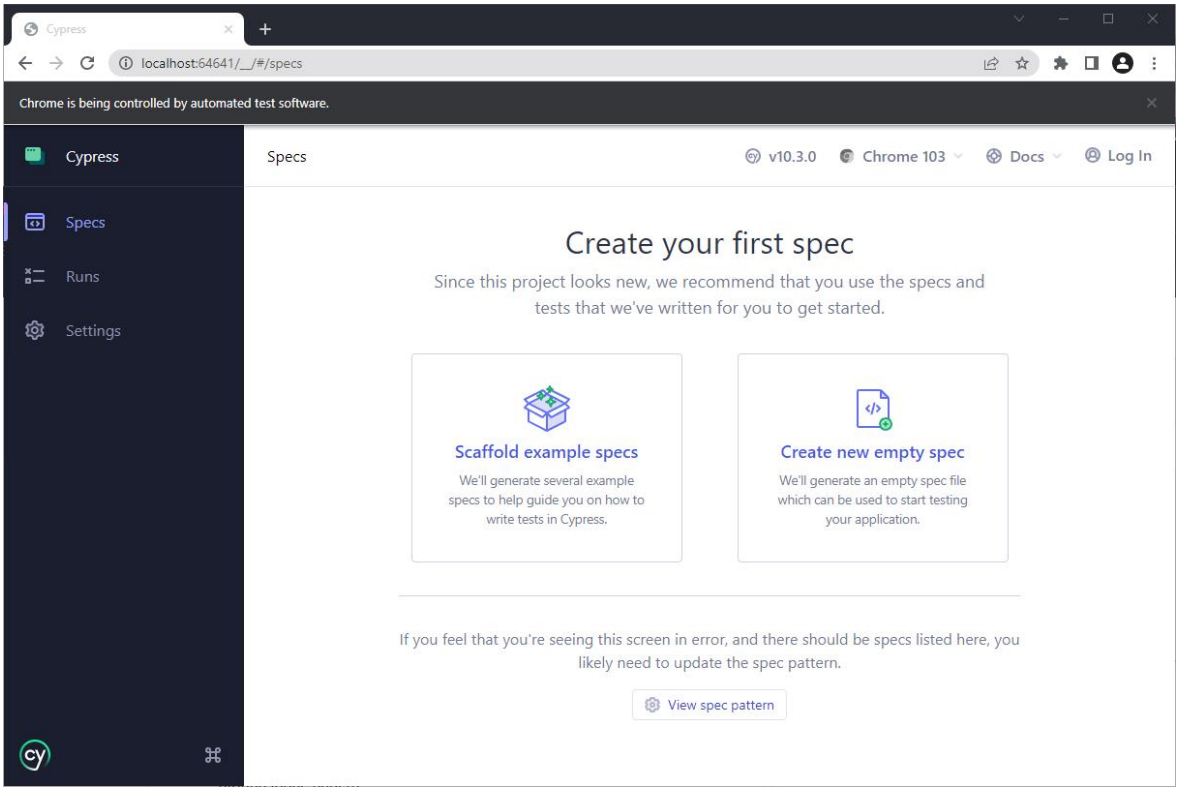
Slika 3: Cypress prozor nakon postavljanja projekta

Nakon što smo postavili projekt, iduće u prozoru nam se prikazuje odabir internetskog preglednika kojeg želimo koristit prilikom testiranja (Slika 4.). Odabrati ćemo *Chrome* internetski preglednik i kliknuti ćemo na „Start E2E testing in Chrome“ botun.



Slika 4: Odabir internetskog preglednika

Nakon što smo odabrali internetski preglednik, primijetit ćemo da nam se otvorio novi prozor i to *Chrome* preglednika (Slika 5.). Iako na prvu izgleda kao obični prozor *Chrome* preglednika, primijetit ćemo da je na vrhu tekst „*Chrome is being controlled by automated test software*.“. Uz to možemo primijetiti i cijelo Cypress korisničko sučelje unutar preglednika.



Slika 5: Chrome prozor nakon odabire internetskog preglednika

Cypress testne datoteke se zovu „*spec*“ datoteke. Ovdje se susrećemo sa pristupačnosti koju Cypress reklamira. Cypress nudi da nam izgenerira primjere testova koji nam mogu koristiti da se upoznamo sa sintaksom i načinom funkcioniranja testova.

6.6. End To End Testovi

U ovome radu pisati ćemo testove za „Moodle“ sustav za podršku nastave na Sveučilišnom odjelu za stručne studije u Splitu. S testovima ćemo pokriti radnje prijave i odjave iz sustava, upis i ispis s e-kolegija, provjeru stranice e-kolegija, izmjenu podataka profila studenta, te provjeru ocjene završnog rada.

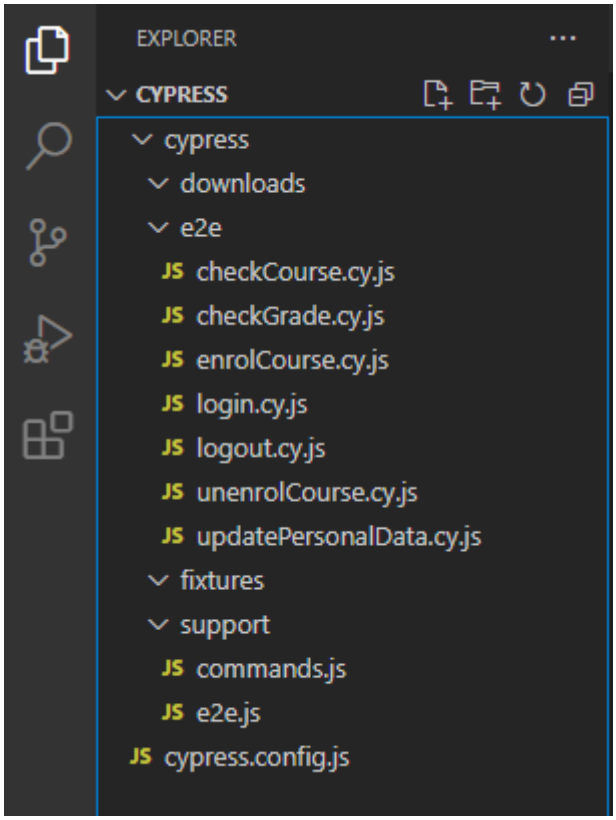
6.6.1. Pisanje testova

Kako bi započeli sa pisanjem testova, potreban nam je uređivač teksta. U ovom slučaju koristiti ćemo „Visual Studio Code“ (u nastavku teksta „VS Code“), globalno popularan uređivač teksta kojeg koriste milijuna programera diljem svijeta.

Otvoriti ćemo „VS Code“, te unutar njega otvoriti prethodno kreiranu mapu Cypress projekta. Kada smo to učinili, unutar „e2e“ mape ćemo kreirati sljedeće datoteke:

- login.cy.js
- logout.cy.js
- enrolCourse.cy.js
- unenrolCourse.cy.js
- checkCourse.cy.js
- updatePersonalData.cy.js
- checkGrade.cy.js

Trebali bi imati sljedeću strukturu mapa i datoteka kao na Slici 6.



Slika 6: Struktura datoteka nakon kreiranja potrebnih testnih datoteka

Sljedeće što trebamo je dodati u „cypress.config.js“ datoteku korisničke podatke za prijavu koji će se koristiti unutar testova. Dodati ćemo dvije varijable pod „env“ (engl. *Enviroment*, okruženje) modul. Varijabla „**studentEmail**“ za elektroničku poštu studenta i „**studentPassword**“ za pripadnu lozinku. Umjesto <studentEmail> i <studentPassword> biti će upisana ispravna elektronička pošta i lozinka. Sadržaj datoteke bi trebao izgledati sljedeće (Ispis 8.):

```
module.exports = {
  env: {
    studentEmail: "<studentEmail>",
    studentPassword: "<studentPassword>",
  },

  e2e: {
    setupNodeEvents(on, config) {
      // implement node event listeners here
    },
  },
};
```

Ispis 8: Sadržaj „cypress.config.js“ datoteke

Sada smo spremni za pisanje testova. Krenuti ćemo od „login.cy.js“, testa koji testira prijavu u sustav za podršku nastavi (Ispis 9.).

```
describe('login into Moodle', () => {
  it('login', () => {
    cy.visit('https://moodle.oss.unist.hr/');

    cy.get('#login_username').type(Cypress.env('studentEmail'));
    cy.get('#login_password').type(Cypress.env('studentPassword'),
        {log: false });

    cy.get('input').contains('Prijava').click();

    cy.get('.logininfo').contains('Prijavljeni ste kao');
  })
})
```

Ispis 9: Sadržaj „login.cy.js“ datoteke

Na ovome testu možemo detaljno pojasniti strukturu samog testa. Svaki test se sastoji od jedne „**describe**“ funkcije koja obuhvaća cijeli test. Prvi parametar funkcije je tekst koji će sadržavati kratki opis testa kako bi nam kasnije olakšao analizu izvršenih testova. Drugi

parametar je funkcija koja sadrži jednu ili više „**it**“ funkcija. Sami koraci testa će se nalaziti unutar „**it**“ funkcija. Prvi parametar je opet tekst koji će sadržavati opis akcije koju radimo. U ovom slučaju za test prijave imamo samo jednu akciju, a to je prijava u sustav. U drugom parametru „**it**“ funkcije pišemo korake testa.

Prvi korak nam je posjećivanje internet stranice samog Moodle sustava za podršku nastavi. Svaki korak započinje sa „**cy**“ prefiksom na kojeg se onda nadovezuju različite naredbe. Kako bi posjetili stranicu Moodle sustava, koristiti ćemo „**visit**“ naredbu koja prima poveznicu internet stranice, u našem slučaju je to „https://moodle.oss.unist.hr/“.

Drugi korak nakon što smo posjetili stranicu nam je unijeti elektroničku poštu i lozinku studenta u odgovarajuća polja. Za to ćemo koristiti dvije naredbe, jedna nadovezana na drugu. Prva je „**get**“ naredba koja ćemo pronaći i dohvatiti „HTML“ element kojeg proslijedimo kao argument naredbe. U našem slučaju tražimo element sa identifikacijskim nazivom „**login_username**“ i „**login_password**“. Ta dva elementa odgovaraju poljima za unos elektroničke pošte i lozinke za prijavu u sustav. Kada samo dohvatili željeni element, možemo upisati tekst u njega (ukoliko element dopušta) s naredbom „**type**“ koja će upisati u element što joj proslijedimo u argumentu same naredbe. U našem slučaju dohvaćamo i prosljeđujemo varijable koje smo prethodno unijeli u konfiguracijsku datoteku samog Cypressa. Tim varijablama pristupamo preko „**Cypress.env**“ naredbe kojoj će argument biti naziv varijable koju želimo dohvatiti. U našem slučaju su to naredbe „**Cypress.env('studentEmail')**“ i „**Cypress.env('studentPassword')**“.

Možemo primijetiti kako prilikom unosa lozinke smo proslijedili i parametar „log: false“. To nam samo omogućava da se taj korak ne prikazuju prilikom izvršavanja testa kako bi lozinka ostala sakrivena.

Treći korak, nakon unosa elektroničke pošte i lozinke, je klik na tipku za prijavu. Kao u prijašnjem koraku, koristiti ćemo naredbu „**get**“ kako bi dohvatili element koji je tipa „**input**“ ali pod uvjetom da sadrži tekst „Prijava“. To postižemo naredbom „**contains**“ koja provjerava da li prethodno dohvaćeni element sadrži tekst koji je proslijeđen u argumentu naredbe. Zatim jedino što nam je preostalo nakon što smo našli odgovarajući botun je kliknuti na njega što postižemo naredbom „**click**“.

Četvrti i posljednji korak je potvrditi da smo se uspješno prijavili u sustav. To postižemo tako da provjerimo da element sa klasom „**logininfo**“ sadrži tekst „Prijavljeni

ste“. Kako je taj tekst vidljiv jedino ukoliko smo uspješno prijavljeni, možemo se na njega osloniti kao provjeru uspješne prijave.

Kako ćemo trebati biti prijavljeni za svaki sljedeći test, umjesto da u svakom testu pišemo opet isti niz naredbe kao u testu za prijavu, možemo kreirati potpuno novu naredbu koju možemo koristiti kao i svaku drugu Cypress naredbu. Kako bi to napravili, unutar „**commands.js**“ datoteke ćemo kreirati novu komandu koja nam izvršava prijavu studenta. Datoteka bi trebala imati slijedeći sadržaj (Ispis 10.).

```
Cypress.Commands.add('loginAsStudent', () => {
    cy.visit('https://moodle.oss.unist.hr/');
    cy.get('#login_username').type(Cypress.env('studentEmail'));
    cy.get('#login_password').type(Cypress.env('studentPassword'), {
log: false });
    cy.get('input').contains('Prijava').click();
});
```

Ispis 10: Sadržaj „commands.js“ datoteke

Prvi parametar funkcije je naziv same funkcije koju kreiramo, te taj naziv ćemo koristiti za pozivanje same funkcije. Jedina razlika između ove funkcije i samog testa prijave je što ova funkcija ne sadrži provjeru. Provjera nije potrebna jer već sami test prijave radi provjeru.

Sljedeći test je „logout.cy.js“ koji testira odjavu iz Moodle sustava (Ispis 11.).

```
describe('logout from Moodle', () => {
  it('logout', () => {
    cy.loginAsStudent();

    cy.get('.logininfo').contains('Odjava').click();

    cy.get('.login').contains('Niste prijavljeni u sustav.');
  })
})
```

Ispis 11: Sadržaj „logout.cy.js“ datoteke

Kao što vidimo, sada možemo koristiti prethodno kreiranu naredbu za prijavu studenta „**cy.loginAsStudent**“, koja će izvršiti cijeli proces prijave studenta u sustav.

Zatim imamo testove koji prijavljuju (Ispis 12.) i ispisuju (Ispis 13.) studenta sa e-kolegija.

```
describe('As a student, enrol a course', () => {
  it('enrols a new course', () => {
    cy.loginAsStudent();

    cy.get('.footer').contains('Svi e-kolegiji').click();
    cy.get('.categoryname').contains('Preddiplomski stručni
studiji').click();
    cy.get('.categoryname').contains('Računarstvo (IT)').click();
    cy.get('.coursename').contains('SRC103 UVOD U
PROGRAMIRANJE').click();
    cy.get('#id_submitbutton').click();

    cy.url().should('eq',
'https://moodle.oss.unist.hr/course/view.php?id=105');
  })
})
```

Ispis 12: Sadržaj „enrolCourse.cy.js“ datoteke

Susrećemo se s dvije nove komande. Prva je „**url**“ koja nam dohvaća trenutnu poveznicu, a druga je „**should**“ koja nam omogućava da radimo razne provjere preko pretpostavki. U ovom slučaju pretpostavljamo da poveznica mora biti jednaka proslijeđenoj poveznici. To nam omogućava da potvrdimo da se nalazimo na ispravnom mjestu na internet stranici, a u ovom slučaju potvrđuje da smo se uspješno prijavili na e-kolegij.

```
describe('As a student, unenrol a course', () => {
  it('unenrols a course', () => {
    cy.loginAsStudent();
    cy.enrolCourse();

    cy.get('a').contains('Ispiši me iz e-kolegija').click();
    cy.get('input').contains('Nastavi').click();


    cy.url().should('eq', 'https://moodle.oss.unist.hr/index.php');
    cy.get('.coursebox').contains('UVOD U
PROGRAMIRANJE').should('not.exist');
  })
})
```

Ispis 13: Sadržaj „unenrolCourse.cy.js“ datoteke

Unutar testa koji ispisuje studenta sa e-kolegija koristimo prethodno spomenutu naredbu „**should**“ koju sad koristimo kako bi potvrdili da e-kolegij nije više prisutan na glavnoj stranici što nam potvrđuje da smo uspješno ispisali studenta s e-kolegija.

Sljedeće testiramo ažuriranje studentskih osobnih podataka (Ispis 14.).

```
describe('As a student, update personal data on Moodle', () => {
    it('updates students data', () => {
      cy.loginAsStudent();

      cy.get('a').contains('Postavke').click({ force: true });
      cy.get('a').contains('Promijeni osobne podatke').click();
      cy.get('a').contains('Opcionalno').click();

      cy.get('#id_url').type('{selectall}https://cypress.io');
      cy.get('#id_phone2').type('{selectall}555-5555');
      cy.get('#id_address').type('{selectall}Adresa 1a, Split');

      cy.get('input').contains('Promijeni osobne podatke').click();

      cy.get('a').contains('Promijeni osobne podatke').click();
      cy.get('a').contains('Opcionalno').click();


      cy.get('#id_url').should('have.value', 'https://cypress.io');
      cy.get('#id_phone2').should('have.value', '555-5555');
      cy.get('#id_address').should('have.value', 'Adresa 1a, Split');
    })
  })
```

Ispis 14: Sadržaj „updatePersonalData.cy.js“ datoteke

Naposljetku imamo test koji provjerava unesenu ocjenu za završni rad (Ispis 15.). Ovaj test je namjerno napravljen da ne završi uspješno i prijavi grešku prilikom izvršavanja. Test neće pronaći odgovarajuću ocjenu (jer još nije unesena) ali će nam poslužiti kao primjer za analiziranje testova i otkrivanje koji je uzrok testa koji nije uspješno završio.

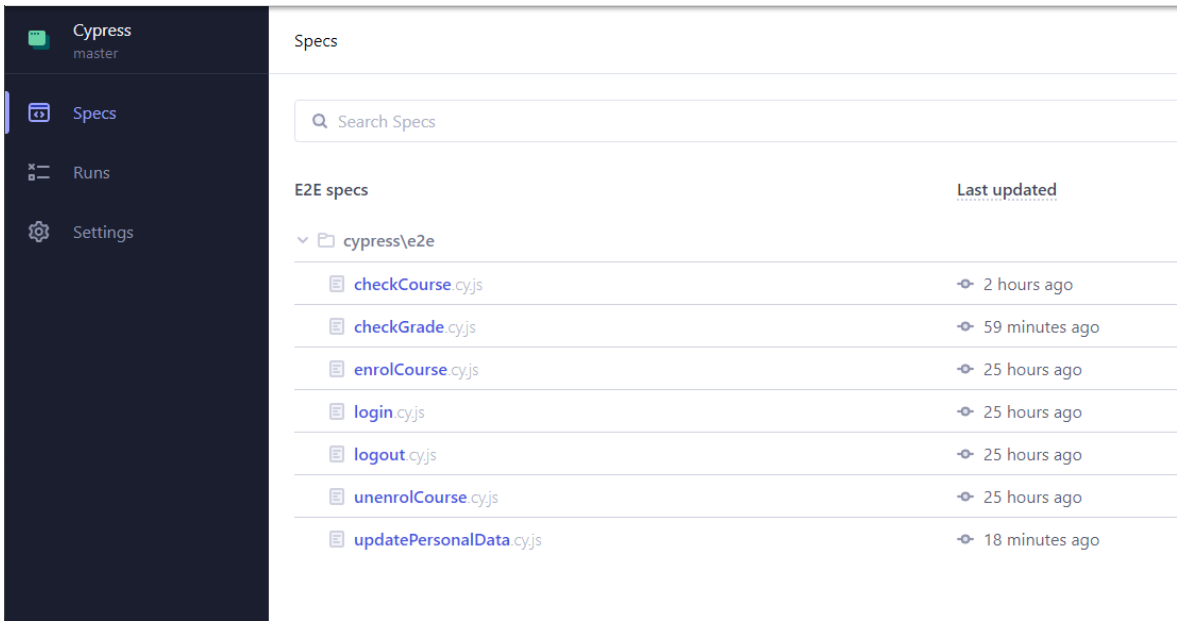
```
describe('As a student, check grade given for Zavrsni rad', () => {
  it('checks if grade for Zavrsni rad is 5', () => {
    cy.loginAsStudent();

    cy.get('a').contains('Ocjene').click({ force: true });
    cy.get('a').contains('ZAVRŠNI RAD').click();

    // this test is supposed to fail until it gets inserted after
successfully completing the thesis.
    cy.get('td').should('have.class', 'column-grade').contains('5');
  })
})
```

Ispis 15: Sadržaj „checkGrade.cy.js“ datoteke

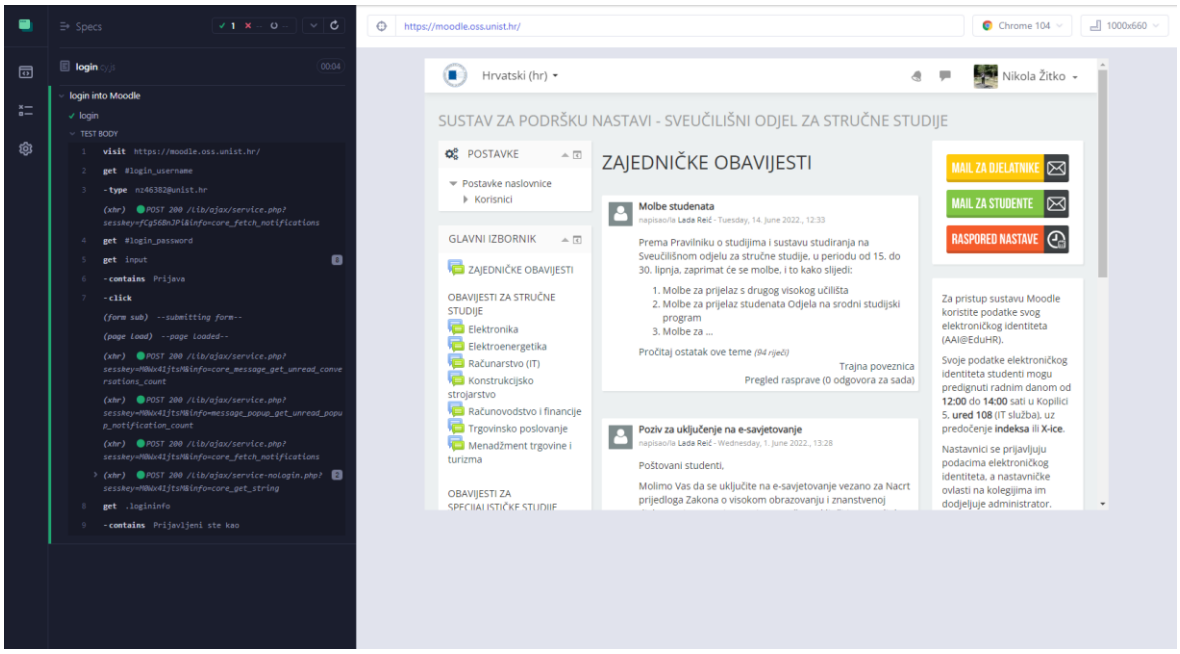
6.6.2. Pokretanje testova

Nakon što smo kreirali testove, sada ih možemo napokon i koristiti. Kako pokrenuli testove, otvoriti ćemo Cypress i odabrati željeni preglednik. Automatski će nam se prikazati popis svih testova koje smo kreirali (Slika 7.). Kako bi pokrenuli određeni test, dovoljno je kliknuti na njega.

Kada kliknemo na određeni test, otvoriti će se novi prozor odabranog preglednika. Prozor će biti podijeljen na dvije polovice. U desnoj polovici ćemo vidjeti preglednik, te u stvarnom vremenu trenutno stanje preglednika kojeg upravlja Cypress. U lijevoj polovici ćemo vidjeti svaki korak testa koji će se izvršiti. Uz korake, tijekom izvršavanja ćemo vidjeti sve rezultate naredbi i raznih poziva koji se odvijaju tijekom izvođenja (Slika 8.).



Slika 7: Cypress testovi spremni za pokretanje



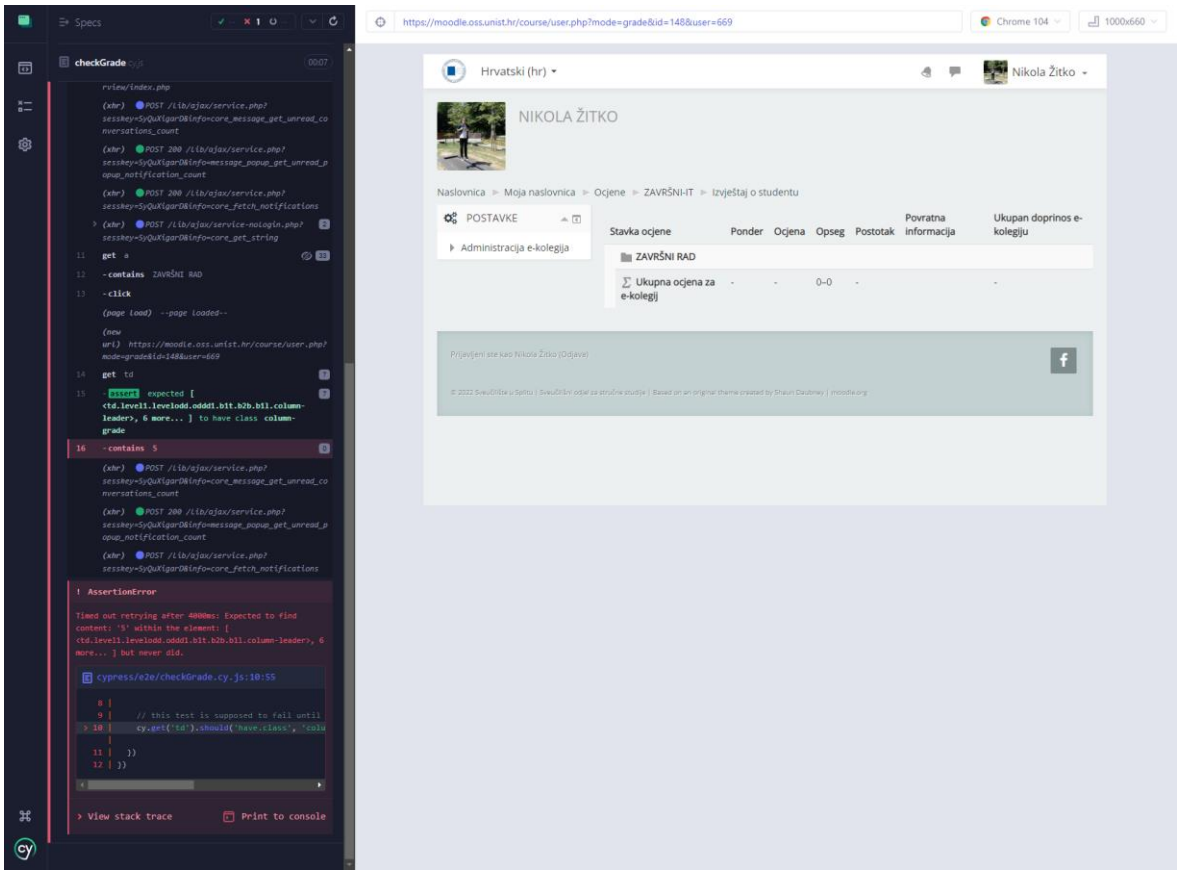
Slika 8: Izvršeni test „login.cy.js“ unutar Chrome preglednika

6.6.3. Analiziranje testova

Tijekom izvršavanja testova u stvarnom vremenu će nam biti prikazano trenutno stanje izvođenja testa, koji se trenutni korak izvodi, koji se pozivi izvršavaju i je li korak uspješno izvršen.

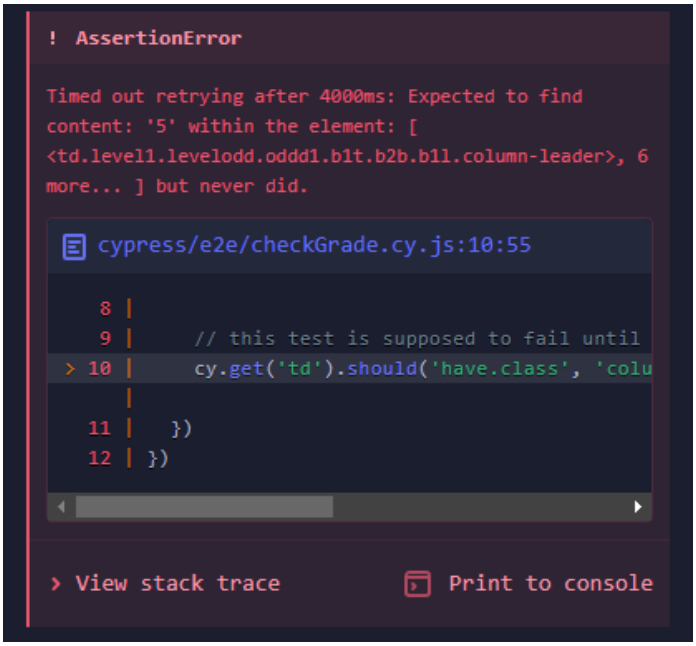
U bilo kojem trenutku je moguće kliknuti na određenu stavku prikazanu u prethodno spomenutoj lijevoj polovici prozora. Ukoliko kliknemo na određenu stavku, prikazati će nam se detaljne informacije koje nam mogu biti korisne prilikom analize testa. Ovu funkcionalnost ćemo rijetko koristiti ukoliko je test uspješno izvršen. Prava korist ove funkcionalnosti je prilikom neuspješno izvršenih testova kako bi otkrili uzrok neuspješnog izvršavanja.

Savršen primjer ovoga možemo vidjeti ukoliko pokrenemo test „checkGrade.cy.js“. Ovaj test je namjerno napisan da se ne izvede uspješno (Slika 9.).



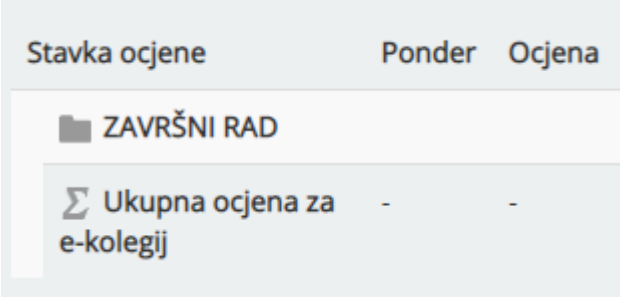
Slika 9: Neuspješno izvršen test "checkGrade.cy.js"

Ukoliko pobliže pogledamo prijavljenu grešku (Slika 10.) vidjeti ćemo da Cypress javlja „**Assertion Error**“. Tu grešku ćemo dobiti ukoliko smo u testu očekivali jednu vrijednost određenog elementa, a Cypress je našao drukčiju vrijednost ili nije uopće našao vrijednost.



Slika 10: "Assertion Error" prilikom izvršavanja "checkGrade.cy.js" testa

U testu smo očekivali da je vrijednost ocjene za završni rad jednaka pet (5), ali Cypress je nije uspio naći. Ukoliko pogledamo trenutno stanje preglednika u trenutku greške, vidjeti ćemo da nedostaje vrijednost pod poljem predviđenom za ocjenu završnog rada (Slika 11.).



Slika 11: Nedostatak ocjene za završni rad

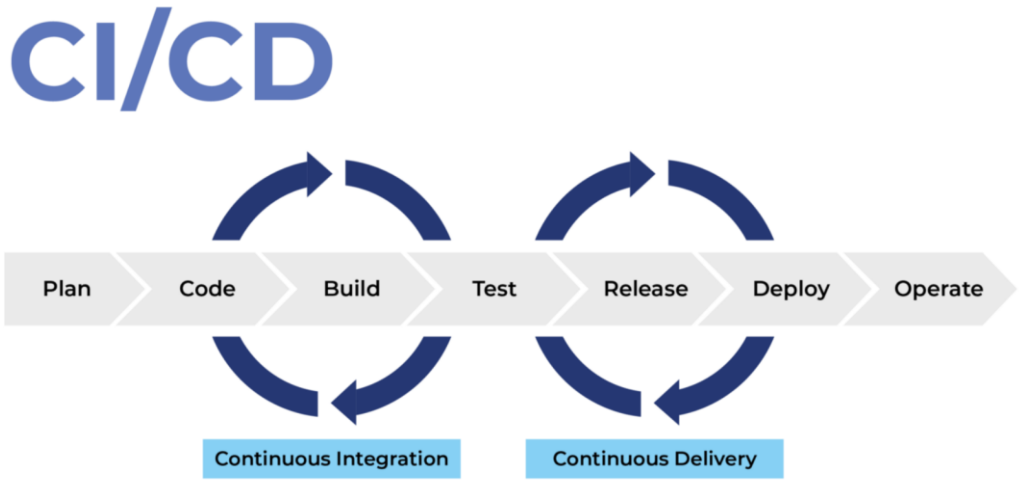
S ovim potvrđujemo da greška nije izazvana pogrešno napisanim testom, već je test uhvatio pogrešku na testiranoj internet stranici.

6.7. Primjer korištenja

U stvarnoj svijetu, u razvojnom procesu modernih internet aplikacija, samo korištenje Cypressa se sitno razlikuje od opisanog u radu. Umjesto samostalnog Cypress projekta, Cypress testovi se direktno integriraju u projekt koji je baziran na jednom od mnogih JavaScript razvojnih okruženja (React, Vue.js, Angular). U tom su slučaju Cypress testovi dio cijelog toka razvoja aplikacije.

U toku kontinuiranog razvoja (Slika 12.) testovi će se automatski pokretati i provjeravati stanje samog projekta kako bi se detektirale i uhvatile pogreške koje mogu utjecati na krajnjeg korisnika kada bi se određena verzija pustila da prođe do samog korisnika.

Moderne alate za verzioniranje (git) je moguće koristiti i za Cypress testove. Omogućavaju popravljanje i ažuriranja testova kako bi ostali u toku sa razvojem ostatka projekta.



Slika 12: Primjer kontinuiranog razvoja

7. Zaključak

U obrazovanju se rijetko spominje testiranje u procesu razvijanja aplikacija ili programa. Većinom je testiranje stavljeno postrance, zanemareno, dok se u fokus stavlja klasično programiranje. U stvarnom svijetu je ipak situacija drukčija. U modernom procesu razvoja aplikacija osiguranje kvalitete (engl. *Quality Assurance*) koje uključuje testiranje je jednako bitno kao i samo razvijanje aplikacije ili programa.

U ovom radu težilo se približiti što je testiranje, čemu služi i zašto je bitno u stvarnom svijetu prilikom izrade aplikacija, te upoznati čitatelje s modernim tehnologijama testiranja. Tema je inspirirana mojim osobnim radom u jednoj modernoj tvrtki za razvoj internet aplikacija. Radom se želi postići osviještenost o bitnom segmentu razvoja aplikacija koji trenutno nije zastupljen u obrazovanju novih stručnjaka informacijskih tehnologija. Također se želi prikazati kako programiranje ne mora biti isključivo razvoj novih značajki već se mogu programirati i automatizirani testovi, što određenoj skupini ljudi može biti zanimljivije od klasičnog programiranja.

Cypress je relativno novi alat, ali nudi odlične mogućnosti, jednostavnost korištenja i odličnu podršku. Pokazao se kao najbolji alat za pisanje automatiziranih testova koji uvelike olakšavaju i ubrzavaju razvoj i održavanje aplikacija. Vrijeme potrošeno na pisanje testova se višestruko vrati samo nakon nekoliko pokretanja jer automatsko izvršavanje je daleko brže, preciznije i točnije od ručnog testiranja aplikacija. Cypress nam daje hrabrost da smo sigurni da aplikacija radi kako treba i nakon napravljenim promjena u kôdu aplikacije.

Literatura

[1] Ehmer, Mohd & Khan, Farmeena. (2012). A Comparative Study of White Box, Black Box and Grey Box Testing Techniques. International Journal of Advanced Computer Science and Applications. 3. 10.14569/IJACSA.2012.030603.

[2] P. Bourque and R.E. Fairley, eds., Guide to the Software Engineering Body of Knowledge, Version 3.0, IEEE Computer Society, 2014.

[3] Kaner, Cem; Bach, James; Pettichord, Bret (2001). Lessons Learned in Software Testing: A Context-Driven Approach.

[4] BrowserStack, „End To End Testing: A Detailed Guide“ https://www.browserstack.com/guide/end-to-end-testing (posjećeno 10.7.2022.)

[5] Brendan Eich, “A Brief History of JavaScript,” July 21, 2010., https://brendaneich.com/2010/07/a-brief-history-of-javascript/ (posjećeno 10.7.2022.)

[6] Flanagan, David (18 April 2011). JavaScript: the definitive guide. Beijing; Farnham: O'Reilly. p. 1. ISBN 978-1-4493-9385-4. OCLC 686709345

[7] Cypress, „JavaScript End to End Testing Framework“, https://www.cypress.io/ (posjećeno 10.7.2022.)