Proces automatiziranog testiranja programske podrške namijenjene autosalonima

Pintek, Lucija

Sveučilište J.J. Strossmayera u Osijeku
Filozofski fakultet Osijek
Dvopredmetni diplomski studij informatologije i informacijskih tehnologija

Lucija Pintek

Proces automatiziranog testiranja programske podrške namijenjene autosalonima

Diplomski rad

Mentor: doc. dr. sc. Tomislav Jakopec

Osijek, 2023.

Sveučilište J.J. Strossmayera u Osijeku
Filozofski fakultet Osijek
Odsjek za informacijske znanosti
Dvopredmetni diplomski studij informatologije i informacijskih tehnologija

Lucija Pintek

Proces automatiziranog testiranja programske podrške namijenjene autosalonima

Diplomski rad

Društvene znanosti, Informacijske i komunikacijske znanosti, Informacijski sustavi i informatologija

Mentor: doc. dr. sc. Tomislav Jakopec

Osijek, 2023.

Prilog: Izjava o akademskoj čestitosti i o suglasnosti za javno objavljivanje

Obveza je studenta da donju Izjavu vlastoručno potpiše i umetne kao treću stranicu završnog odnosno diplomskog rada.

IZJAVA

Izjavljujem s punom materijalnom i moralnom odgovornošću da sam ovaj rad samostalno napravio te da u njemu nema kopiranih ili prepisanih dijelova teksta tuđih radova, a da nisu označeni kao citati s napisanim izvorom odakle su preneseni.
Svojim vlastoručnim potpisom potvrđujem da sam suglasan da Filozofski fakultet Osijek trajno pohrani i javno objavi ovaj moj rad u internetskoj bazi završnih i diplomskih radova knjižnice Filozofskog fakulteta Osijek, knjižnice Sveučilišta Josipa Jurja Strossmayera u Osijeku i Nacionalne i sveučilišne knjižnice u Zagrebu.

U Osijeku, datum 26.8.2023.

Lucija Pintek, 0122228777
ime i prezime studenta, JMBAG

Zahvala

Veliko hvala Valentini Valentić, stručnoj mentorici, na pomaganju kroz cijeli proces pisanja diplomskog rada. Mentorica koja razumije moj način razmišljanja i učenja te pažljivo bira način na koji će mi nesebično prenijeti svoje znanje. Mentorica koja na svaki problem ne daje rješenje nego navodi ka njemu kako bih naučila doći sama do istog. Pravi mentor koji se rijetko pronalazi.

Posebno se zahvaljujem za odvojeno vrijeme u trenutcima kada je bilo najviše posla, na pomoći prilikom odabira teme, svim savjetima, preporukama, podršci i razumijevanju. Iz sveg srca, ovog juniorskog, hvala!

Sažetak

Svrha ovog diplomskog rada pobliže je objasniti automatizirano testiranje, kreirati automatizirane testove za programsku podršku namijenjenu autosalonima i implementirati u okviru stvarnog projekta. Kako bi se razumjelo automatizirano testiranje, na početku je objašnjeno ručno testiranje i kratka povijest testiranja. Jira i Xray alati, kao i Cypress posebno su prikazani kao odabrani alati i razvojno okruženje za implementaciju automatiziranih testova. Organizacija projekta pojednostavljuje proces automatizacije te je kao takva predstavljena. Naknadno, s organizacijom direktorija i dokumenata, opisan je dizajn rješenja i programsko rješenje. U nastavku, prikazani su automatizirani testovi i objašnjeni koraci unutar definiranih testova usporedno sa snimkom zaslona onoga što se testira. Naposljetku, objašnjeno je na koji se način automatizirani testovi mogu izvršiti pomoću Cypress korisničkog sučelja koristeći se naredbama unutar terminala integriranog razvojnog okruženja. Kraj procesa automatizacije označava analiza izvještaja rezultata, prijavljivanje bugova i daljnje planiranje. Pisanje automatiziranih testova u Cypress-u donosi izazove poput organizacije testova kako bi što duže ostali dosljedni i mogli jednostavno nadopunjavati i izmjenjivati, a tu malim dijelom pomažu .json datoteke te funkcije koje mogu generirati više testova unutar jednog. Zatim asinkronost gdje je jedino bilo potrebno shvatiti da Cypress jako puno radnji izvršava u pozadini jedne naredbe. Potom *iframe* koji se mogao dohvatiti dodavanjem "{"chromeWebSecurity": false}" u cypress.json datoteku. Naposljetku, izazov je održavati testove i svakim novim sprintom nadopunjavati novim testovima.

Ključne riječi: Cypress, automatizirano testiranje, automatizirani testovi, QA, razvojna okruženja

Sadržaj

1. Uvod 1
2. Osnove automatiziranog testiranja 4
3. Razvojno okruženje i alati 12
   3.1. Cypress 12
   1.2. Sustav za praćenje aktivnosti projekta 16
   1.3. Xray Test Management 17
4. Testiranje programske podrške namijenjene autosalonima 19
5. Izazovi 41
6. Zaključak 43
Literatura 44
Prilozi 46

1. Uvod

Smatra se kako se testiranjem započelo usporedno s razvojem programske podrške (engl. *software*) u 20. stoljeću, nedugo nakon drugog svjetskog rada. Tom Kilburn, računalni znanstvenik, napisao je dio programske podrške 1948. godine koji je izvodio matematičke izračune. Glavni oblik testiranja u to vrijeme bilo je otklanjanje pogrešaka (engl. *debugging*) te se kao oblik zadržao sve do 80-ih godina 20. stoljeća. Razvojni timovi tih godina razmatrali su testiranje na višoj razini, kao što je testiranje aplikacija u stvarnim okruženjima te definiranje testiranja kao procesa osiguranja kvalitete, a činio je dio životnog ciklusa razvoja programske podrške. Samo 10 godina kasnije, testiranje je od jednog dijela životnog ciklusa razvoja programske podrške postalo sveobuhvatan proces koji se zove osiguranje kvalitete, a pokriva cijeli životni ciklus. Kao sveobuhvatan proces, utječe na sve ostale procese unutar životnog ciklusa razvoja programske podrške.[1]

Testiranje se u određenim definicijama može opisati kroz različite dijelove. Prema International Software Testing Qualifications Board (ISTQB) dijeli se na testiranje kao proces i ciljeve procesa testiranja. Proces testiranja nije jednoličan te uključuje sve aktivnosti životnog ciklusa, statičko i dinamičko testiranje, planiranje, pripremu i evaluaciju. Kroz navedene procese može se interpretirati važnost i zahtjevnost testiranja, a ciljevi su procesa utvrđivanje zadovoljava li programska podrška propisane zahtjeve, odgovara li programska podrška svrsi te otkrivanje nedostataka. Testiranje pomaže izmjeriti kvalitetu programske podrške kroz broj pronađenih pogrešaka (engl. *bug*) izvođenjem testova. Na taj način, pronalaskom nedostataka povećava se kvaliteta sustava kada se isti poprave, pod uvjetom da su popravljeni ispravno.[2] Navedeni ciljevi ujedno su i glavni razlog testiranja, ukoliko programska podrška zadovoljava propisane zahtjeve, njegova kvaliteta je osigurana. Identificiranje ne ispunjenih zahtjeva, problema upotrebljivosti i funkcionalnosti osigurava poboljšanje korisničkog iskustva popravkom takvih grešaka. Važno je naglasiti kako se testiranjem izbjegava rizik od pravnih i financijskih kazni jer se testiranjem pomaže osigurati usklađenost programske podrške sa standardima i propisima industrije.[3] Ušteda troškova svakako je jedan od važnijih razloga testiranja, budući da kasniji pronalazak pogreške u

[1] Usp. IBM. URL: https://www.ibm.com/topics/software-testing (2023-07-03)

[2] Usp. Graham, Dorothy…[et. al.]. Foundations of software testing: ISTQB Certification. Australia…[et.al]: Cengage Learning Emea, 2008., str. 13.

[3] Usp. Software Testing Fundamentals. URL: https://softwaretestingfundamentals.com/ (2023-07-03)

životnom ciklusu označava skuplji popravak. Zato je važno uključiti testere u ranije faze životnog ciklusa gdje se bug može pronaći već u fazi definiranja zahtjeva programske podrške.[4] Naposljetku, testiranje poboljšava reputaciju tvrtke osiguravanjem visoke kvalitete programske podrške. Posljedice netestiranja programske podrške mogu biti manje poput kašnjenja, uzaludno potrošenog vremena, ali i izrazito velike poput gubitka novca, gubitka poslovnog ugleda pa i smrt ovisno o vrsti programske podrške koji se testira.[5] Ovakvo definiranje testiranja prikladno je za bilo koju razinu testiranja s malim razlikama u njihovim ciljevima.[6]

Ukratko opisano, testiranje odrađuje Quality Assurance (QA) tim, koji može uključivati QA inženjera, QA arhitekta, QA analitičara, QA menadžera, QA voditelja tima ili druge pozicije, ovisno o veličini projekta. Posao QA inženjera obuhvaća testiranje, identificiranje problema, analiziranje bugova tijekom testiranja, izrada izvještaja, predlaganje rješenja za poboljšanje programske podrške te pomoć developeru pri poboljšanju. Odlikuje se u vještinama poput stručnosti, pažnji posvećenoj detaljima, sposobnost rješavanja problema te komunikacijske sposobnosti. QA analitičar stvara, implementira i bilježi temeljite testne strategije, pokreće testove, locira suvišne elemente, izrađuje i održava operativne postupke te dokumentaciju. Može se reći da QA inženjer ovisi o QA analitičaru koji organizira informacije koje bi mu pomogle u radu. Od vještina svakako zahtijeva logičku analizu i sposobnost organiziranja informacija, izradu tehničke dokumentacije, stručnost u radu s velikim količinama podataka, želju za istraživanjem, kao i komunikacijske sposobnosti. Nadalje, QA arhitekt izrađuje odgovarajuće test planove, metodologije, u tijeku je s najnovijim pristupima u polju automatizacije. Njegove su vještine razumijevanje skriptnih jezika i poznavanje najboljih praksi, izgradnja radnih okruženja za automatizaciju, testiranje vrlo kompliciranih sustava, te kao svi prethodno navedeni, dobre komunikacijske sposobnosti. QA menadžer izrađuje plan rada za sve ostale u timu, utvrđuje mjere, predviđa troškove rada te prati cijeli proces testiranja. Samo neke od vještina su komunikativnost, rješavanje složenih problema, analitičko modeliranje te sposobnost upravljanja vremenom. QA voditelj tima prati napredak tima, procjenjuje učinkovitost i nadgleda procese. Vještine potrebne za ovu poziciju su razumijevanje različitih strategija, proučavanje zahtjeva prije dizajniranja i osnovno razumijevanje programskih i označiteljskih jezika. Često jedna osoba obavlja više poslova, no svaka pozicija ima određenu važnost.[7]

---

[4] Usp. Graham, Dorothy…[et. al.]. Nav. dj., str. 13.
[5] Usp. Software Testing Fundamentals. URL: https://softwaretestingfundamentals.com/ (2023-07-03)
[6] Usp. Graham, Dorothy…[et. al.]. Nav. dj., str. 13.
[7] Usp. Safonova, Elena.Understanding the core positions in QA team and ways to hire them, 2022. URL: https://sumatosoft.com/blog/understanding-the-core-positions-in-qa-team-and-ways-to-hire-them (2023-07-03)

Međutim, glavna je podjela između ručnih i automatiziranih testera. Ručno testiranje najbolje je za pronalaženje grešaka koje su povezane s osnovnom poslovnom logikom aplikacije. Poslovna logika je kod koji implementira korisničke zahtjeve, složena je i zahtjeva čovjeka koji će provjeriti njezinu točnost te se smatra kako automatizacija u nekim slučajevima nije prikladna. Takvo testiranje posebno ističe svoju važnost zbog velike količine varijacija, različitih scenarija i previše mogućih neuspjeha za automatizaciju da sve to prati.[8] Dok automatizirano testiranje obuhvaća sve vrste testiranja uključujući kompatibilnost više platformi i neovisna je o procesu, vrste testova poput funkcionalnih, performanse, regresijski i ostali, mogu se automatizirati. Automatizirano testiranje pripada razvoju programske podrške, ali ne zamjenjuje potrebu za ručnim testiranjem i testerima jer sve njihove vještine služe kao uvjet za automatizirano testiranje.[9]

Diplomski rad sastoji se od teorijskog i praktičnog dijela. Cilj rada pobliže je objasniti što su automatiziranih testovi, važnost istih te kreirati regresijske automatizirane testove za programsku podršku namijenjenu autosalonima u sklopu stvarnog projekta. Prvo poglavlje opisuje vrste automatiziranog testiranja s naglaskom na regresijsko automatizirano testiranje. Opisuje proces implementacije automatizacije u projekt, koji su aktualni alati za automatiziranje te njihove kratke opise. Drugo poglavlje podijeljeno je na tri potpoglavlja. Prvo opisuje Cypress okruženje, primjer End-to-end (E2E) testiranja, kako se pišu naredbe, tvrdnje i testovi. Drugo opisuje Jira alat, koji predstavlja sustav za praćenje aktivnosti projekta. U trećem se opisuje Xray alat koji kroz Jira alat donosi rezultate Cypress testova. Također je prikazano na koji se način izvoze rezultati unutar Cypress-a, te uvoze u Jiru putem integriranog Xray alata. Četvrto poglavlje predstavlja praktični dio diplomskog rada, u kojemu je prikazana struktura direktorija i datoteka, a zatim opisana programska podrška za koju su kreirani automatizirani testovi. Automatizirani testovi prikazani su usporedno s korisničkim sučeljem koje testiraju te detaljno objašnjeni. U petom poglavlju navedeni su izazovi na koje se naišlo tijekom kreiranja automatiziranih testova i na koji su način razriješeni. Naposljetku je izveden zaključak na osnovu rada.

[8] Usp. Whittaker, James A. Exploratory software testing. Boston: Addison-Wesley, 2009. Str. 14-15.
[9] Usp. Dustin, Elfriede; Garrett, Thom; Gauf, Bernie. Implementing automated software testing. Boston: Addison-Wesley, 2009. Str. 4.

2. Osnove automatiziranog testiranja

Kao što je navedeno u uvodnom dijelu, sva praksa, tehnika i znanje ručnog testiranja isprepletena je s automatiziranim testiranjem. Učinkovita implementacija automatizacije nije samo testiranje sučelja, niti je ograničeno na određenu fazu testiranja. Na različitim sustavima izvode se različite vrste testiranja, od testiranja sigurnosti, preko testiranja performansi do testiranja funkcionalnosti. Sve vrste testiranja u pravilu pripadaju jednoj od glavnih kategorija: testiranje crne kutije (engl. *black-box testing*), testiranje bijele kutije (engl. *white-box testing*) s testiranjem sive kutije (engl. *gray-box testing*) kao međukategorija. Testiranje bijele kutije uključuje vrste testiranja kao što je jedinično testiranje (engl. *unit testing*), što znači da je bilo potrebno nešto znati o kodu i dizajnu za vrijeme testiranja. S druge strane, testiranje crne kutije ne zahtijeva poznavanje nikakvih unutarnjih procesa jer se testovi izvode pozivanjem različitih funkcionalnosti kroz interakciju korisničkog sučelja. S obzirom da takvo testiranje neće otkriti sve nedostatke, pri testiranju crne kutije primjenjuje se testiranje sive kutije kako bi se povećala učinkovitost.[10] Testiranje sive kutije uključuje kombinaciju testiranje bijele kutije i crne kutije, što znači da je unutarnja struktura djelomično poznata testeru. Primjerice, poznat je pristup internim strukturama podataka i algoritmi u svrhu dizajniranja testnih slučajeva.[11] Pri tome, tijekom testiranja sive kutije, tester može biti precizniji u testiranju onih dijelova programske podrške koja su sklonija pogreškama zbog visokog stupnja složenosti ili zbog nestabilnosti uzrokovane novim kodom.[12] Iako je moguće automatizirati sve testove, neki nisu vrijedni automatizacije, stoga je važno dobro procijeniti što automatizirati. Testiranje sigurnosti (engl. *security testing*), pokrivenosti koda (engl. *code coverage testing*), paralelno testiranje (engl. *concurrency testing*) te ostali slični testovi prilikom ručnog testiranja skloni su pogreškama, postaju nedosljedni, a rezultiraju gubitkom vremena, novca i napora. Stoga je takve testove bolje izvoditi u automatiziranom okruženju koje pruža prednost poput ponovljivosti. Shodno tome, testovi koji se uglavnom uvijek procjenjuju važnima za automatizaciju su:

- jedinično testiranje (engl. *unit testing*)
- regresijsko testiranje (engl. *regression testing*)
- funkcionalno testiranje (engl. *functional testing*)

[10] Usp. Dustin, Elfriede; Garrett, Thom; Gauf, Bernie. Nav. dj., str. 5-13.
[11] Usp. Gray box testing: software testing. URL: https://www.geeksforgeeks.org/gray-box-testing-software-testing/ (2023-07-05)
[12] Usp. Dustin, Elfriede; Garrett, Thom; Gauf, Bernie. Nav. dj., str. 5-13.

- testiranje otpornosti na stres (engl. *stress testing*)
- testiranje izvedbe (engl. *performance testing*)
- sigurnosno testiranje (engl. *security testing*)
- testiranje pokrivenosti koda (engl. *code coverage testing*).[13]

Automatizirano testiranje ključna je tehnika kojom se rješavaju izazovi na koje testeri nailaze. Ukoliko je automatizacija dobro implementirana, ona smanjuje vrijeme i troškove testiranja, poboljšava kvalitetu programske podrške, povećava pokrivenost testiranja te zamjenjuje svakodnevne radno intenzivne zadatke ručnog testiranja.[14] Sve navedeno omogućava testeru da posveti svoje vrijeme drugim zadacima, poslovnom razvoju, dokumentaciji i sl.

Za potrebe razumijevanja rada, pobliže će se objasniti automatizirano regresijsko testiranje koje predstavlja jedan od prvih koraka pri implementaciji automatizacije unutar projekata. Ono se provodi nakon što se napravi promjena unutar koda, bilo to zbog nove značajke, uklonjenih grešaka (engl. *debugging*) ili ostalih mogućnosti. Često se nakon nekih promjena unutar jednog dijela koda utječe na drugi dio, gdje razvojni programer (engl. *developer*) nenamjerno oštećuje drugu značajku. Stoga, nakon svakog novog postavljanja (engl. *deployment*), tester ponovno testira cijelu aplikaciju i njezino izvođenje važnih značajki kako bi se spriječile nove greške na produkciji i neželjene reakcije korisnika.[15] Produkcija je završna faza proizvodnje aplikacije gdje je ona kao takva distribuirana javnosti te ju koriste javni korisnici.[16] Osobnosti regresijskog testiranja kao što su ponovljivost i dugotrajnost, dovoljan su razlog za njihovu automatizaciju. U automatiziranom obliku mogu se ponavljati svakodnevno bez velike ljudske intervencije u manjem vremenskom razdoblju. Također je moguća povećana pokrivenost testiranja različitih preglednika istim testnim slučajevima. Kako bi se započelo s automatizacijom regresijskih testova, potrebno je proći kroz više slojeva procesa. Na početku, važno je razumjeti zahtjeve aplikacije koja se testira te koje vrste testova

[13] Isto, str. 15.
[14] Isto, str. 23.
[15] Usp. Automated regression testing: a detailed guide, 2023. URL: https://www.browserstack.com/guide/automated-regression-testing (2023-07-07)
[16] Usp. QA testing: what is DEV, SIT, UAT and PROD?, 2020. URL: https://medium.com/@buttertechn/qa-testing-what-is-dev-sit-uat-prod-ac97965ce4f (2023-07-07)

su potrebne. Zatim se odabire alat i razvojno okruženje za automatizirano testiranje, primjerice Cypress za testiranje korisničkog sučelja. Alternative u odnosu na Cypress su:

- Selenium
- Nightwatch.js
- Puppeteer
- Playwright
- itd.

Selenium omogućuje automatizaciju mrežnog preglednika kroz jednostavnu interakciju s HTML elementima. Koristi Chrome DevTools Protocol (DCP) za kreiranje lažnih mrežnih zahtjeva i odgovora. Ima mogućnost istovremenog testiranja u više preglednika i mogućnost upravljanja karticama u pregledniku.

Nightwatch.js okvir je za automatizaciju izgrađen na Node.js platformi. Njegove su značajke paralelno testiranje u više okruženja, izvršavanje testova i generiranje izvješća jednom naredbom.

Puppeteer je okvir koji omogućuje automatizirano testiranje Google Chrome preglednika, također za većinu zadataka poput rukovanje zahtjevima i odgovorima te lociranje predmeta koristi Node.js. Kao značajke ističu se generiranje snimki zaslona i PDF-ovi mrežnih stranica, te presretanje i modificiranje mrežnih zahtjeva kako bi se uklonile greške na mreži.[17]

Playwright se kao i Cypress temelji na JavaScriptu, iako podržava i druge programske jezike. Podržava izvođenje testova na više preglednika i korisničkog konteksta istovremeno. Omogućava stvaranje instanci objekta kako bi se istovremeno pokrenulo više kartica, preglednika i sl.[18]

Za potrebe diplomskog rada, odabran je Cypress jer se na stvarnom projektu Cypress koristi kao zadani alat za automatizirano testiranje. Zbog prirode posla, jednog zadanog mrežnog preglednika te dobrog omjera kvalitete i kvantitete, odabran je Cypress.

Nakon odabira razvojnog okruženja, popisuju se slučajevi koji će se automatizirati, uzimajući u obzir kako je većina slučajeva podložna automatizaciji, te se započinje s pisanjem automatiziranih testova pomoću odabranog radnog okruženja. Pri kraju pisanja automatiziranih

[17] Usp. Katalon. URL: https://katalon.com/resources-center/blog/end-to-end-e2e-testing-tools-frameworks (2023-07-24)
[18] Usp. Chako, Oleksandr. Playwright vs Cypress: which framework to choose for E2E testing?. URL: https://eleks.com/research/playwright-vs-cypress/ (2023-07-24)

testova, oni se pokreću lokalno kako bi se utvrdilo postoje li potencijalne lažne greške (engl. *false failures*) koje mogu dovesti do nepotrebnih prijava. Takvi testovi smatraju se nepouzdanima te je važno pri samom početku otkriti bilo kakve nedostatke i popraviti ih. Nakon što su testovi pouzdani i spremni za izvođenje, važno ih je integrirati u alate koji će moći izvještavati rezultate testova nakon što su izvršeni. Naposljetku, cilj svake automatizacije je pokretanje istih kroz CI/CD alate gdje se testovi mogu izvoditi svakodnevno ili prilikom postavljanja nove verzije u QA okruženje.[19] Koraci su isti za implementaciju bilo koje vrste testova.

Autori Dustin, Garrett i Gauf u svojoj knjizi napravili su nekoliko tablica za izračun uštede vremena i novca automatiziranim testiranjem. Tablica 1. prikazuje rezultate ukupne uštede vremena na temelju zbroja radnih listova: ušteda vremenskog postavljanja testnog okruženja, ušteda vremena razvoja testova, ušteda vremena izvršavanja testova i ušteda vremena evaluacije testova. Ušteda je u tablici iskazana u satima. Svaki navedeni dio uštede u tablici računa se pomoću zasebnog radnog listića. Primjerice, ušteda pri postavljanju testnog okruženja uključuje različite aktivnosti kao što su:

- postavljanje testnog okruženja,
- ponovno postavljanje baze,
- ugradnja komponenti,
- provjera konfiguracije i sl.

S obzirom da se ovaj postupak može automatizirati, ali nije isti svaki put tj. ovisi o odabiru okruženja (Cypress, Selenium itd.), autori su stoga naveli "xxx" u polju jer nemaju približno točan podatak. Nadalje, ušteda vremena razvoja testova podijeljena je u više kategorija, a vrijednosti u zagradama prikazuju brojke autora na temelju kojih su napravljeni izračuni u prikazanoj tablici:

- koliko je testova planirano (ručno → 1000, automatizirano → 1000),
- procijenjeno vrijeme kreiranja jednog testa u minutama (ručno → 15, automatizirano - 30),
- ukupno vremena potrebno za kreiranje testova u satima (ručno → 250, automatizirano → 500).

[19] Usp. Automated regression testing: a detailed guide, 2023. URL: https://www.browserstack.com/guide/automated-regression-testing (2023-07-25)

Ušteda vremena izvršavanja testova podijeljena je u 4 kategorije, a vrijednosti u zagradama prikazuju brojke autora na temelju kojih su napravljeni izračuni u prikazanoj tablici:

- koliko je testova planirano (ručno → 1000, automatizirano → 1000),
- procijenjeno vrijeme za izvođenje jednog testa u minutama (ručno → 10, automatizirano → 0,5),
- procijenjen broj ponavljanja testova (ručno → 10, automatizirano → 10),
- ukupno vrijeme procijenjeno za izvođenje testova u satima (ručno → 1666,66, automatizirano → 83,33).

Za računanje procijenjenog vremena izvođenja jednog testa u minutama za automatizirane testove, formula je: vrijeme za ručno/20.

Posljednje, ušteda vremena evaluacije rezultata testova podijeljena je također u 4 kategorije, a vrijednosti u zagradama prikazuju brojke autora na temelju kojih su napravljeni izračuni u prikazanoj tablici:

- broj rezultata testova za analizirati (ručno → 3000, automatizirano → 3000),
- procijenjeno vrijeme za evaluaciju jednog rezultata testa u minutama (ručno → 5, automatizirano → 0,5),
- procijenjen broj ponavljanja testova (ručno → 10, automatizirano → 10),
- ukupno vrijeme za evaluaciju rezultata testova u satima (ručno → 2500, automatizirano → 250).

Za računanje procijenjenog vremena za evaluaciju jednog testa u minutama za automatizirane testove, formula je: vrijeme za ručno/10.

Tako su autori dobili izračun od 3683,33 sati uštede vremena na automatizaciji 1000 testova.[20]

[20] Usp. Dustin, Elfriede; Garrett, Thom; Gauf, Bernie. Nav. dj., str. 57-62.

Tablica 1. Prikaz izračuna uštede implementacijom automatiziranog testiranja

Radni list ukupne uštede automatizacije	
Zadaci	**Vremenska ušteda automatizacije (sati)**
Ušteda vremena pri postavljanju testnog okruženja	xxx
Ušteda vremena razvoja testova	-250
Ušteda vremena izvršavanja testova	1583,33
Ušteda vremena evaluacije rezultata testova	2250
Sati	3583,33
Dani (8 sati)	447,9
Mjeseci (20 dana)	22,4

U Tablici 2. prikazan je izračun uštede vremena kreiranjem testova na projektu koji su prikazani u radu, ali i kreiranjem onih koji nisu prikazani jer su iste funkcije i naredbe već objašnjene u prikazanim testovima, a odnose se na iste funkcionalnosti. Važno je uzeti u obzir kako u ovaj izračun ulazi mali broj testova koji se izrađuju unutar jednog sprinta, stoga brojke neće biti značajno velike.

Izračun uštede vremena pri postavljanju testnog okruženja:

- postavljanje okruženja poput baze, alata i sl. u satima (ručno → 2h, automatizirano → 1h)

Izračun uštede vremena razvoja testova:

- koliko je testova planirano (ručno → 90, automatizirano → 90),
- procijenjeno vrijeme kreiranja jednog testa u minutama (ručno → 15, automatizirano - 30),

- ukupno vremena potrebno za kreiranje testova u satima (ručno → 22,5, automatizirano → 45).

Izračun uštede vremena izvršavanja testova:

- koliko je testova planirano (ručno → 90, automatizirano →90),
- procijenjeno vrijeme za izvođenje jednog testa u minutama (ručno → 10, automatizirano → 0,5),
- procijenjen broj ponavljanja testova (ručno → 2, automatizirano → 2),
- ukupno vrijeme procijenjeno za izvođenje testova u satima (ručno → 30, automatizirano → 1,5).

Izračun uštede vremena evaluacije rezultata testova:

- broj rezultata testova za analizirati (ručno → 180, automatizirano → 180),
- procijenjeno vrijeme za evaluaciju jednog rezultata testa u minutama (ručno → 5, automatizirano → 0,5),
- procijenjen broj ponavljanja testova (ručno → 2, automatizirano → 2),
- ukupno vrijeme za evaluaciju rezultata testova u satima (ručno → 30, automatizirano → 3).

Tablica 2. Prikaz izračuna uštede implementacijom automatiziranog testiranja na primjeru diplomskog rada

Radni list ukupne uštede automatizacije	
Zadaci	**Vremenska ušteda automatizacije (sati)**
Ušteda vremena pri postavljanju testnog okruženja	1
Ušteda vremena razvoja testova	-22,5
Ušteda vremena izvršavanja testova	28,5
Ušteda vremena evaluacije rezultata testova	27
Sati	34
Dani (8 sati)	4,3
Mjeseci (20 dana)	0,2

Svakako je važno naglasiti i rizike automatiziranog testiranja i razumjeti sve pretpostavke i preduvjete te ih predstaviti kao dio poslovnog slučaja. Ono što to uključuje su svi događaji i okolnosti koji mogu spriječiti uspješnu implementaciju ili izvršavanje automatiziranih testova. To mogu biti stalne promjene zahtjeva, nedostupnost programske podrške, nedostatak vještine, kasno odobrenje proračuna i sl. Svaki rizik potrebno je opisati te ponuditi način na koji bi se on mogao smanjiti ili u potpunosti spriječiti. Iako je automatizacija testnih slučajeva poželjna i sadrži prednosti u odnosu na neke procese ručnog testiranja, ona može biti neuspješna. Neuspjeh se definira u prekoračenju troškova, neotkrivenim kritičnim softverskim pogreškama, odstupanja u rasporedu, prekoračenju troškova i ostalih negativnih strana loše implementacije automatiziranog testiranja.[21]

[21] Usp. Isto, str. 65.

3. Razvojno okruženje i alati

3.1. Cypress

Brian Mann osnovao je Cypress 2014. godine s idejom revolucioniziranja testiranja postavljajući Cypress izravno u preglednik i na taj način dijelom stvarnog vremena izrade aplikacije u agilnom okruženju. Neke od poznatih tvrtki koje koriste Cypress u svakodnevnom razvoju su Slack, Netflix, NBA, Disney i ostali.[22] Cypress je alat za testiranje namijenjen koji osigurava pisanje bržih, jednostavnijih i pouzdanijih testova za razliku od drugih takvih alata. Omogućuje pisanje različitih vrsta testova i može testirati sve ono što radi unutar preglednika, no ovim radom opisat će se End-to-end testiranje. End-to-end, poznati kao i E2E testovi, posjećuju aplikaciju u pregledniku i izvode radnje putem korisničkog sučelja kao što radnje izvodi i krajnji korisnik.[23] Sukladno službenoj dokumentaciji, ono što su u implementaciji u programskom jeziku metode u nastavku će se nazivati naredbama. Na Slici 1. nalazi se prikaz jednostavnog E2E testa koji obuhvaća testiranje većeg dio koda na klijentu i na poslužitelju.

```
74  describe( title: 'Objava', fn: () => {
75      it( title: 'Kreiranje nove objave', fn: () => {
76          cy.visit('/posts/new')
77
78          cy.get("input.post-title")
79              .type( text: "Moja prva objava");
80
81          cy.get("input.post-body")
82              .type( text: "Pozdrav, svijete!");
83
84          cy.contains("Objavi")
85              .click();
86
87          cy.get("h1")
88              .should( chainer: "contain", value: "Moja prva objava");
89      });
90  });
```

Slika 1. Prikaz jednostavnog E2E testa

[22] Usp. Cypress. URL: https://www.cypress.io/about-us/our-story/ (2023-07-25)
[23] Usp. Cypress. URL: https://docs.cypress.io/guides/overview/why-cypress (2023-07-25)

Cypress koristi jQuery kako bi testovi bili poznati i čitljivi razvojnim developerima, no postoji razlika u dohvaćanju i pristupanju Objektnom modelu dokumenta (engl. *Document Model Object*), tj. DOM elementima. Razlika se nalazi u tome što Cypres za sve DOM upite obavlja logiku ponovnih pokušaja pronalaska elementa i vremenskog ograničenja pronalaska. Takva logika bolje odgovara načinu rada mrežnih aplikacija te osigurava manje nestabilnih testova. Moguće je lociranje elementa pomoću sadržaja koji korisnik vidi, a dohvaća se pomoću naredbe cy.contains(), na Slici 2. prikazano je dohvaćanje elementa klase "main" koji u sadržaju ima tekst "Nova objava". Lociranje je moguće izvesti samo pomoću cy.contains() naredbe, međutim dodavanje klase elementa ili bilo kojeg drugog obilježja osigurava preciznije lociranje.

```
90    cy.get('.main').contains( content: 'Nova objava')
```

Slika 2. Prikaz dohvaćanja elementa prema sadržaju

Testovi unutar Cypress-a ne padaju istog trenutka kada element nije pronađen, upravo njegova asinkronost daje aplikaciji vremenski okvir kako bi izvršilo sve što radi. To se izvršava pomoću timeout opcije, a vremensko ograničenje može se konfigurirati globalno ili na temelju naredbe. U primjeru na Slici 3., test daje elementu 10 sekundi kako bi se pojavio, u suprotnom on pada.

```
59    cy.get('.my-slow-selector', { timeout: 10000 })
```

Slika 3. Prikaz vremenskog ograničenja

Interakcija s elementima ide kroz ulančavanje naredbi, rezultat izvođenja svake naredbe daje ulaz sljedećoj naredbi sve dok se ne izvrše ili se ne pojavi greška. Na Slici 4. prikazan je primjer gdje se prvo .click() ulančava na .contains(), a on na cy.get(), govoreći mu da klikne na DOM element koji sadrži određeni tekst, dok u drugom dijelu .type() ulančava na .get() govoreći mu da upiše predmet dobiven iz .get() upita.

```
90    cy.get('.text-editor-button').contains( content: 'uredi').click()
91        .get( selector: 'textarea.post-body').type( text: 'Dobra objava.')
```

Slika 4. Prikaz ulančavanja naredbi

Neki od primjera naredbi koje Cypress pruža za interakciju s aplikacijom:

- .focus() - stavlja fokus na DOM element
- .clear() - brisanje vrijednosti unosa
- .select() - odabire <option> unutar <select>
- .dblclick() - klikne dvaput na DOM element.

Dodavanjem takvih naredbi, osigurava se točnost o stanju u kojem je element prije izvođenja ostalih radnji. Primjerice naredba .click() osigurava komunikaciju s elementom, Cypress automatski čeka aktivno stanje elementa, a ono znači da element nije skriven, onemogućen i sl.

Tvrdnje (engl. *assertions*) također osiguravaju da se opiše željeno stanje elementa, primjer na Slici 5. osigurava da je "checkbox" element vidljiv, "input" ima određenu vrijednost, a "form" CSS klasu. Takve tvrdnje, Cypress automatski čeka sve dok elementi ne dostignu navedeno stanje, u suprotnom takvi testovi padaju.

```
67  cy.get(':checkbox').should( chainer: 'not.be.disabled')
68
69  cy.get('input').should( chainer: 'have.value',  value: 'HRV')
70
71  cy.get('form').should( chainer: 'have.class',  value: 'form-one')
```

Slika 5. Prikaz tvrdnji

Za određene naredbe, nije potrebno dodavati tvrdnje kako bi osigurali stanje elementa jer su tvrdnje ugrađene u samu naredbu. Primjerice:

- cy.visti() - očekuje status kod stranice 200
- cy.request() - očekuje postojanost udaljenog servera koji osigurava odgovor
- cy.get() - očekuje da element postoji unutar DOM
- .type() - da je element u stanju za tipkanje
- itd.

Kao što je već navedeno, Cypress naredbe ne rade ništa u trenutku kada se pozivaju, nego se stavljaju u red kako bi se kasnije pokrenule. Dakle, naredbe se izvode unutar određenog redoslijeda.

```
100 ▶ it( title: 'obriše tekst nakon što se očisti element', fn: () => {
101       cy.visit('/my/resource/path') // 1.
102
103       cy.get('.text-box') // 2
104           .should( chainers: 'have.text') // 3
105           .clear() // 4
106
107       cy.get('.text-box') // 5
108           .should( chainers: 'not.have.text') // 6
109   });
```

Slika 6. Primjer za izvršavanje naredbi redoslijedom

U primjeru na Slici 6., navedeni test bi se izvršio sljedećim redoslijedom:

1. posjeti URL
2. pronađi element s klasom .text-box
3. utvrdi ima li element tekst
4. izvrši radnju čišćenja sadržaja iz elementa
5. pronađi element s klasom .text-box
6. utvrdi da element nema više tekst.

Navedene radnje uvijek se izvršavaju jedna za drugom, razlog k tomu su pozadinske radnje koje Cypress čini. Osim što posjeti URL, Cypress pričeka da se učitaju svi vanjski izvori bez dodatnih naredbi. Naredba cy.visit() za razliku od drugih naredbi čeka dulje prije nego što istekne te ima svoju posebnu vrijednost vremenskog ograničenja definiranu u konfiguraciji. U drugom i petom koraku, pronalazi element i ponavlja naredbu sve dok nije pronađen ili dok ne istekne vremensko ograničenje. U trećem i šestom koraku, osim što osigurava tvrdnju elementa, u ovom slučaju da tekst postoji i ne postoji, također ponavlja naredbu dok tvrdnja nije postignuta. U četvrtom koraku, izvršava radnju tek nakon što pričeka da element dosegne stanje u kojem se može izvršiti navedena radnja. Cypress na taj način radi dodatan posao kako bi osigurao da stanje aplikacije odgovara onome što naredbe očekuju od iste. Iako se većina naredbi izvršava identično svaki puta, neke naredbe mijenjaju stanje preglednika, na primjer:

- cy.request() - automatski dobiva i postavlja kolačiće na udaljeni poslužitelj

- cy.clearCookies() - uklanja sve kolačiće iz preglednika
- click() - aplikacija reagira na klik.[24]

1.2. Sustav za praćenje aktivnosti projekta

Kako bi aktivnosti unutar tima bile dobro organizirane i raspoređene, potrebno je koristiti neki od sustava za praćenje aktivnosti. Stoga je unutar ovog rada odabran Jira sustav. Jira predstavlja cijeli niz proizvoda unutar jednog paketa. Svaki od proizvoda predstavlja agilno rješenje za upravljanje poslom što pospješuje suradnju unutar tima.[25] Također predstavlja alat koji se najčešće koristi u agilnim timovima, a omogućuje planiranje kroz različite tipove zadataka kao što su: developer story, test story, bug, test, test set, test plan i sl. Ovisno o željama i radu tima, tipovi zadataka mogu biti različiti te nisu isti za svaki projekt.[26] Unutar Jire može se koristiti Scrum ploča i Kanban ploča, a razlikuju se u tome što je Scrum ploča okvir za upravljanje projektima koji pomaže timu strukturirati i upravljati radom, dok se pomoću Kanban ploče oslanja na vizualne zadatke za upravljanje projektima.[27] Neki od važnih pojmova za Scrum ploču su sprint, backlog, user story i epic. Sprint je kratki vremenski ograničeno razdoblje unutar kojeg tim ima cilj završiti zadanu količinu posla, najčešće traje 2 tjedna. Backlog je nešto poput zaostatka, a njime upravlja vlasnik proizvoda (engl. *product owner*) tj. PO. Unutar backlog-a nalaze se svi zadaci za razvojni tim koji su izvedeni iz plana, primjerice neki bug-ovi, poboljšanja ili značajke koje je tek potrebno provesti, a svaka stavka ima rang, opis, procjenu i vrijednost. Korisnička priča (engl. *User story*) općenito objašnjava značajke nove funkcionalnosti ili programske podrške u cijelosti iz perspektive korisnika, a svrha mu je izraziti kako bi značajka ili programska podrška trebala izgledati. Naposljetku, epic predstavlja velik opus rada jedne ili više značajki, velika korisnička priča koja je podijeljena na nekoliko manjih, a potrebno je više sprinteva kako bi se završio jedan epic.[28] Za Kanban izdvojit će se pojmovi ograničenja rada u tijeku, tijek rada i točka isporuke. Ograničenja rada u tijeku (engl. *Work in progress limits*) kritična su za otkrivanje situacija koje sprječavaju napredak zadataka i maksimiziranje tijeka rada (engl. *workflow*). Tijek rada predstavlja stupce na ploči koji

[24] Usp. Cypress. URL: https://docs.cypress.io/guides/core-concepts/introduction-to-cypress (2023-07-25)
[25] Usp. Atlassian. URL: https://www.atlassian.com/software/jira/guides/more/jira-family#what-is-the-jira-family (2023-07-25)
[26] Usp. Atlassian. URL: https://www.atlassian.com/software/jira?tab=plan (2023-07-25)
[27] Usp. Atlassian. URL: https://www.atlassian.com/agile/kanban/kanban-vs-scrum (2023-07-25)
[28] Usp. Atlassian. URL: https://www.atlassian.com/software/jira/features/scrum-boards (2023-07-25)

predstavljaju određene aktivnosti poput onih “za napraviti”, “u tijeku”, “gotovo” i sl. Točka isporuke (engl. *delivery point*) kraj je tijeka rada i tada je programsko rješenje spremno za kupca.[29] Sa stajališta testera, Jira omogućava jednostavno pisanje bug izvještaja koji se dodjeljuju razvojnom programeru i pomoću tijeka rada na ploči jednostavno prati faza u kojoj se bug izvještaj nalazi. Jira kao alat, omogućava spajanje s drugim alatima poput Xray Test Management for Jira. Kroz navedene alate, privodi se kraj cijelom procesu automatiziranih testova prikazom rezultata izvršenih testova. Kroz spajanje takvih alata, pojednostavljuje se rad testera i razvojnih programera na način da je sve dostupno na jednom mjesto te svatko iz tima u bilo kojem trenutku ima pristup određenim dijelovima projekta. U narednom odlomku, prikazan je način na koji Jira i Xray prikazuju rezultate izvršenih automatiziranih testova.

1.3. Xray Test Management

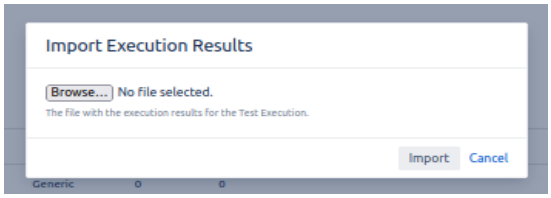
Pomoću Xray Test Management alata integriranim unutar Jira alata, omogućava se jednostavan tijek rada za testere. Pomoću Xray-a, kreiraju se test planovi, izvršavaju testovi na različitim okruženjima, organiziraju se testovi unutar repozitorija, a naposljetku i rezultati rada testera kroz izvješća o broju bugova, defecta i sl.[30] Kao što se izvode ručni testovi kroz Xray te je vidljiv rezultat izvedenog testa unutar svakog test plana, to je potrebno napraviti i za automatizirane testove. Osim što korisničko sučelje Cypress-a vizualno prikazuje rezultate izvedenih testova, potrebno ih je nekako implementirati unutar Jire kako bi bili vidljivi cijelom timu. Svaka specifikacijska datoteka obrađuje se zasebno tijekom svakog izvođenja Cypress testova. Unutar Cypress konfiguracije dodaje se dio koda (Slika 7.) koji stvara zasebne .xml datoteke u mapi rezultata. Svako novo izvođenje testova zamjenjuje prethodno izvješće novim, a kako bi se sačuvalo jedinstveno izvješće svaki put, koristi se [hash] u nazivu.[31]

[29] Usp. Atlassian. URL: https://www.atlassian.com/software/jira/features/kanban-boards (2023-07-25)
[30] Usp. Atlassian. URL: https://marketplace.atlassian.com/apps/1211769/xray-test-management-for-jira?tab=overview&hosting=cloud (2023-07-25)
[31] Usp. Cypress. URL: https://docs.cypress.io/guides/tooling/reporters (2023-07-25)

```
import { defineConfig } from 'cypress';
export default defineConfig( config: {
  experimentalMemoryManagement: false,
  reporter: 'junit',
  reporterOptions: {
    mochaFile: 'results/my-test-output-[hash].xml',
    toConsole: true,
  },
```

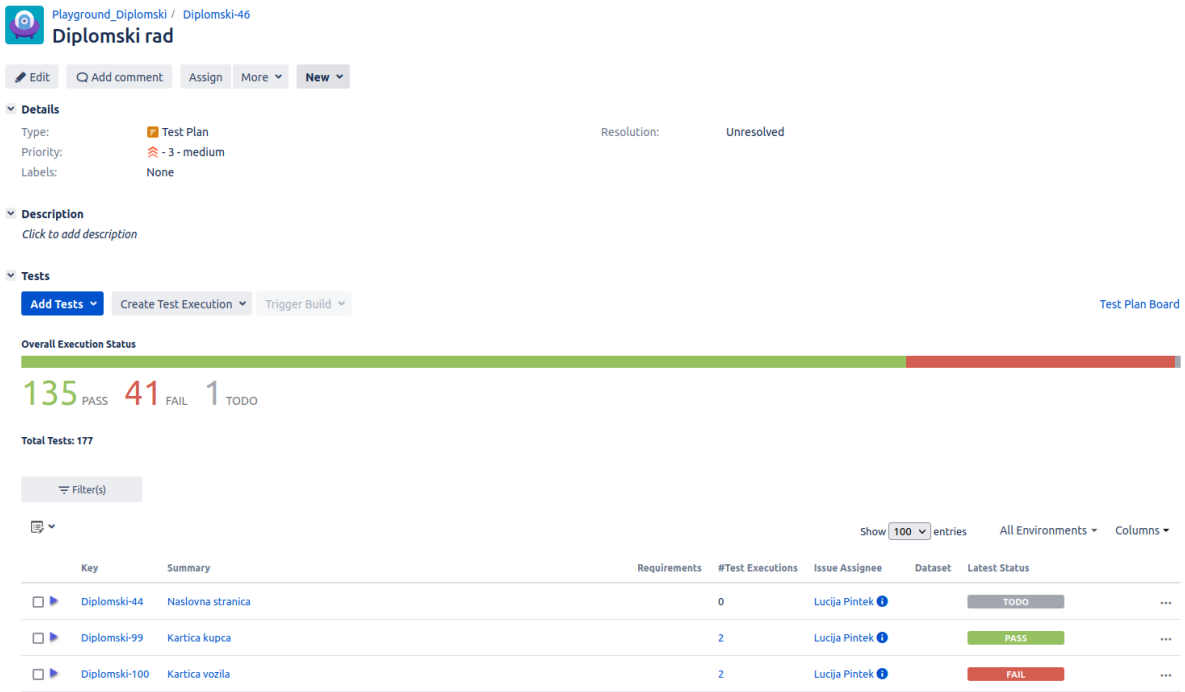
Slika 7. Prikaz konfiguracije za izvoz .xml datoteke sa rezultatima izvedenih testova

Kroz uvoz .xml datoteke unutar Jira alata (Slika 8.), dostupni su rezultati izvršenih testova dostupni svim članovima tima.



Slika 8. Uvoz .xml datoteke unutar Jira alata

Nakon što je .xml datoteka učitana, Slika 9. pokazuje kako ono izgleda unutar Jira alata.



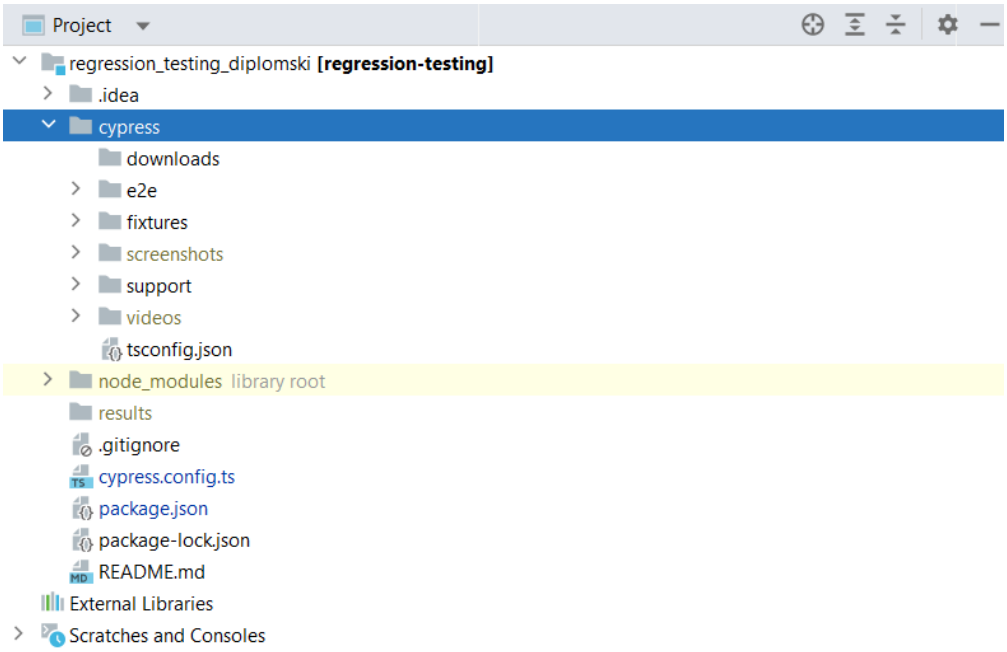
Slika 9. Prikaz rezultata učitanih u Jira alat

4. Testiranje programske podrške namijenjene autosalonima

Izrađeni testni slučajevi i implementacija osmišljeni su za stvarni projekt. Sve funkcije, podaci, nazivi, izgled aplikacije i ostalo što će se koristiti je modificirano za potrebe diplomskog rada kako bi se poštivala sigurnost i privatnost podataka. Struktura i organizacija podataka ostaje ista, s obzirom da je izrađena prema već postojećim praksama testiranja u Cypress alatu.

Za izradu Cypress automatiziranih testova odabrano je IntelliJ IDEA integrirano razvojno okruženje (engl. *Integrated Development Environment*), a Typescript kao programski jezik. Razlog tomu je standardna praksa na stvarnom projektu, međutim bilo koje drugo integrirano razvojno okruženje može se koristiti za izradu automatiziranih testova u Cypress alatu. Na Slici 10. prikazana je struktura projekta u IntelliJ IDE alatu. Unutar jednog glavnog projekta naziva "regression_testing_diplomski" nalaze se direktoriji koji su važni za E2E testiranje. Svaki projekt ima sljedeće direktorije:

- .idea
- cypress
    - downloads
    - e2e
    - fixtures
    - screenshots
    - support
    - videos
- node_modules
- results.



Slika 10. Izgled projekta u IntelliJ IDEA

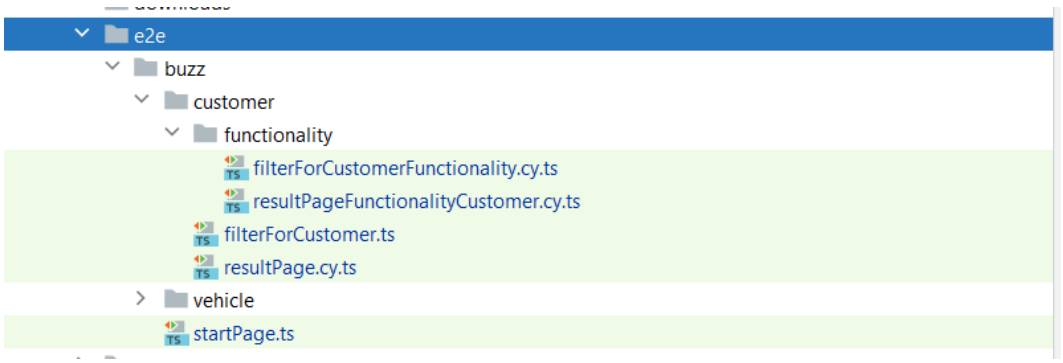
Najvažniji podaci za konfiguraciju nalaze se u datoteci "cypress.config.ts", podaci unutar "env:" važni su za automatizirani test prilikom prijave. Upisivanjem korisničkih podataka u ovu datoteku, prilikom podizanja projekta dovoljno je obrisati podatke samo na navedenom mjestu kako oni ne bi bili otkriveni od neželjenih strana. (Slika 11.)

cypress.config.ts

```
import { defineConfig } from 'cypress';
export default defineConfig( config: {
  experimentalMemoryManagement: false,
  reporter: 'junit',
  reporterOptions: {
    mochaFile: 'results/my-test-output-[hash].xml',
    toConsole: true,
  },
  env: {
    buzzBaseUrl: 'exampleBuzz@url.com',
    username: 'student1',
    password: 'sTuDeNt1'
  },
  e2e: {
    viewportWidth: 1920,
    viewportHeight: 1080,
    defaultCommandTimeout: 12000,
    watchForFileChanges: false
  }
})
```

Slika 11. Prikaz konfiguracije unutar Cypress-a

U “e2e” direktoriju nalaze se svi testovi organizirani unutar više drugih direktorija. Na ovom primjeru je e2e > buzz (naziv aplikacije) > customer (dio aplikacije koji obuhvaća) > functionality (koje testove obuhvaća). Svaka spremljena datoteka napisanih testova završava nastavkom “cy.ts”. (Slika 12.)

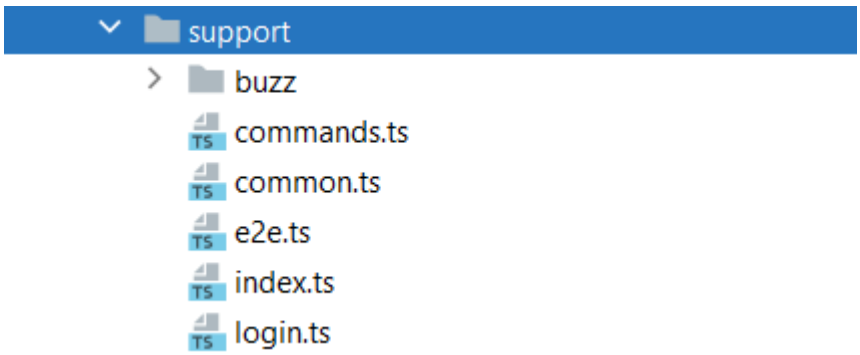


Slika 12. Sadržaj e2e direktorija

“Fixtures” direktorij također je organiziran u više drugih direktorija, ovisno za što su datoteke namijenjene. U navedenom direktoriju nalaze se sve .json datoteke, u njima su popisani svi elementi i njihovi id-jevi, klase i sl. selektori prema kojima će se tražiti na stranici. Svrha .json datoteke su efikasnost i efektivnost, popisivanjem selektora elemenata u jednoj datoteci osigurava jednostavnije prepravke u slučaju promjena u kodu. Na taj način u testovima ništa nije tvrdo kodirano (engl. *hard coded*) što dovodi do kvalitetnijih automatiziranih testova. (Slika 13.)

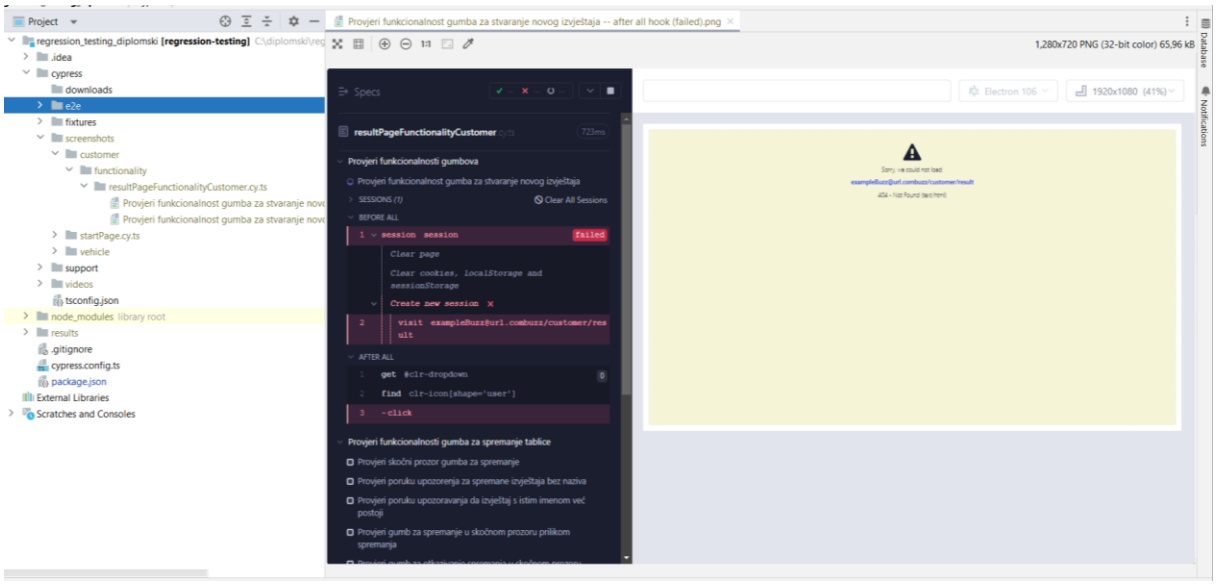
Slika 13. Sadržaj fixtures direktorija

Direktorij “support” sadrži sve one datoteke u kojima su napisane naredbe i funkcije koje se koriste unutar testova. (Slika 14.)



Slika 14. Sadržaj support direktorija

Cypress također ima funkcionalnost gdje napravi snimku zaslonu kada jedan od testova padne, upravo te snimke spremaju se u “screenshots” direktorij. Svaka spremljena snimka može se otvoriti u bilo kojem trenutku dok se ne izbriše, a na prikazu je otvorena unutar IDE alata gdje prikazuje problem o krivo napisanoj poveznici na stranicu. (Slika 15.)



Slika 15. Prikaz funkcionalnosti snimke zaslona

Prije izrade automatiziranih testova izrađuju se testni planovi za ručno testiranje, način na koji će se prolaziti kroz aplikaciju i što se od nje očekuje. Na samom početku izrađen je dizajn rješenja (engl. *Solution design*) koji prije svega donosi informaciju kako mrežna aplikacija treba izgledati uz popratnu dokumentaciju koja navodi sve zahtjeve u mrežnoj

aplikaciji. Iščitavanjem dokumentacije, izrađuju se i provode testovi za ručno testiranje. Mrežna aplikacija može sadržavati funkcionalnosti koje su zahtjevnije za automatizirano testiranje, kao što je izvoz .xlsx datoteka i provjeravanje točnosti podataka te općenita provjera točnosti podataka unutar tablica. Prema tome, u ovom primjeru sve funkcionalnosti koje se nalaze u mrežnoj aplikaciji dio su automatiziranog testiranja, osim provjera točnosti podataka unutar tablica. Za taj je dio u početku odlučeno testiranje s ručnim testovima i izvođenjem SQL skripti u bazi podataka, iako je i to moguće automatizirati. U nastavku je prikazana izrada automatiziranih testova na primjeru dizajna rješenja.

Aplikacija namijenjena autosalonima, ovdje imenovana Buzz, ima svoje funkcionalnosti za koje je potrebno napraviti automatizirane regresijske testove, a podijeljena je u nekoliko stranica. Svrha ove aplikacije je omogućiti autosalonima pregled svih kupaca i vozila koja su na stanju ili prodana i sl. Prije svega potrebno je prijaviti se sa svojim podacima kako bi se pristupilo mrežnoj aplikaciji. (Slika 16.) S lijeve strane je prikaz korisničkog sučelja, a s desne strane prikaz funkcije login() koja se koristi u naredbi za početak svake nove sesije testova, dakle izvodi se na početku izvođenja testova kako bi se napravila prijava koja je primarna za nastavak testiranja. Funkcija je napisana u "login.ts" datoteci, a elemente dohvaća preko "login.json" datoteke unutar koje su definirani selektori te podatke za prijavu dohvaća iz konfiguracije pomoću "env" varijable.

Slika 16. Prikaz prijave u mrežnu aplikaciju

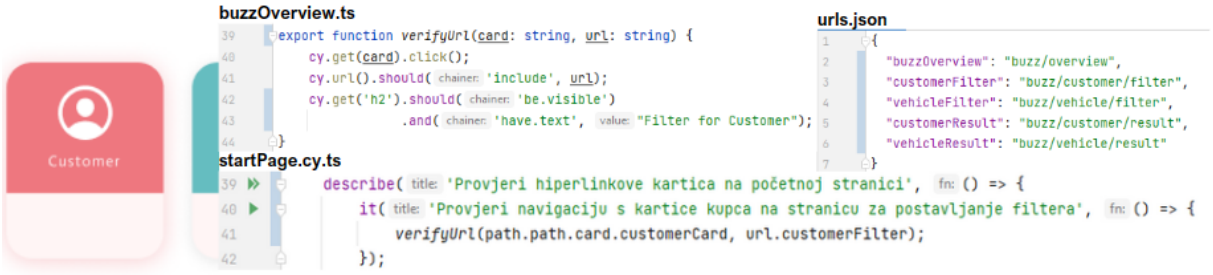
Unutar svake datoteke u kojoj su pisani testovi, definirani su before() hook i beforeEach() hook. Before() hook izvodi se jednom prije nego se započne s izvođenjem testova.

U primjeru na Slici 17. before() hook koristi se kako bi se jednom prije svih testova otvorio određeni URL stranice. S druge strane, beforeEach() hook koristi se kada se želi izolirati testove tako da ne utječu jedni na druge i ono se izvodi prije svakog testnog slučaja. Na primjeru aplikacije, before() hook posjećuje URL stranicu jednom, dok beforeEach() prije svakog testa čeka da se stranica učita na što indicira "spinner". Primjer je iz testnog dokumenta stranice sa tablicom, dok se isti nalaze s manjim izmjenama unutar drugih testnih dokumenata.

Slika 17. Prikaz before() i beforeEach() hook-ova

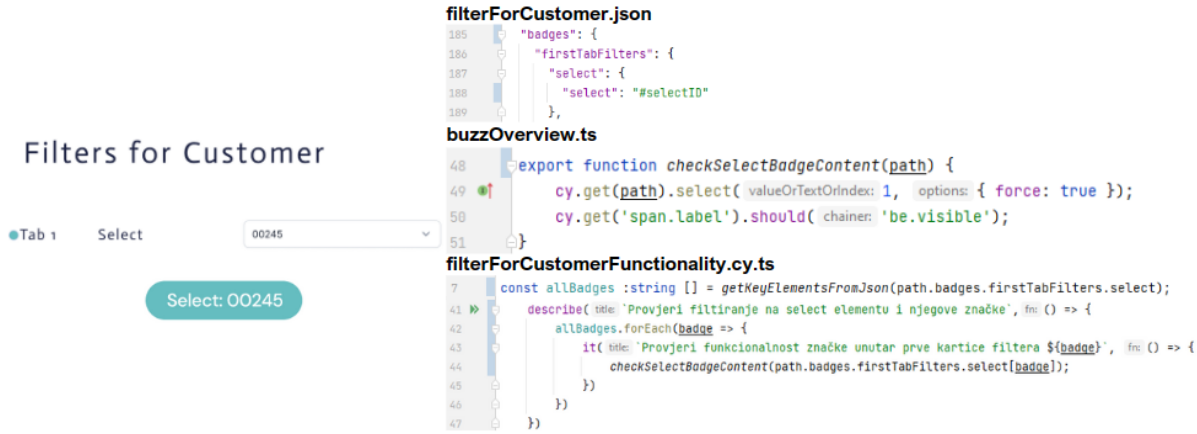
U nastavku će biti prikazani testovi koji čine jedan proces korisnika kada dođe na mrežnu aplikaciju. Naslovna stranica sadrži kartice na koje se može kliknuti, a ovisno o odabiru kartice i filtriranja, dobiva se pojednostavljeni prikaz podataka korisnika autosalona. U ovom primjeru, nalaze se dvije kartice gdje jedna predstavlja kupca, a druga sva prijevozna sredstva unutar autosalona. S obzirom na isti proces unutar aplikacije kroz obje kartice, prikazat će se testovi samo na primjeru kartice kupca. Aplikacija omogućava kreiranje izvještaja, stoga je

sljedeći korak korisnika odabir kartice unutar koje želi napraviti izvještaj. Prvo je potrebno provjeriti vodi li klik na karticu do stranice za postavljanje filtera. Unutar "buzzOverview.ts" datoteke, kreirana je funkcija "VerifyUrl" koja zahtijeva definiranje kartice i URL-a. Dohvaća karticu i izvršava klik na nju, zatim dohvaća URL koji mora biti jednak definiranom URL-u, naposljetku element "h2" treba biti vidljiv i imati naslov "Filter for Criteria" na novootvorenoj stranici. Nakon što se klikne na jednu od kartica s naslovne stranice, korisniku se otvara stranica na kojoj filtrira željene podatke. Ti koraci definirani su funkcijom "VerifyUrl". (Slika 18.)



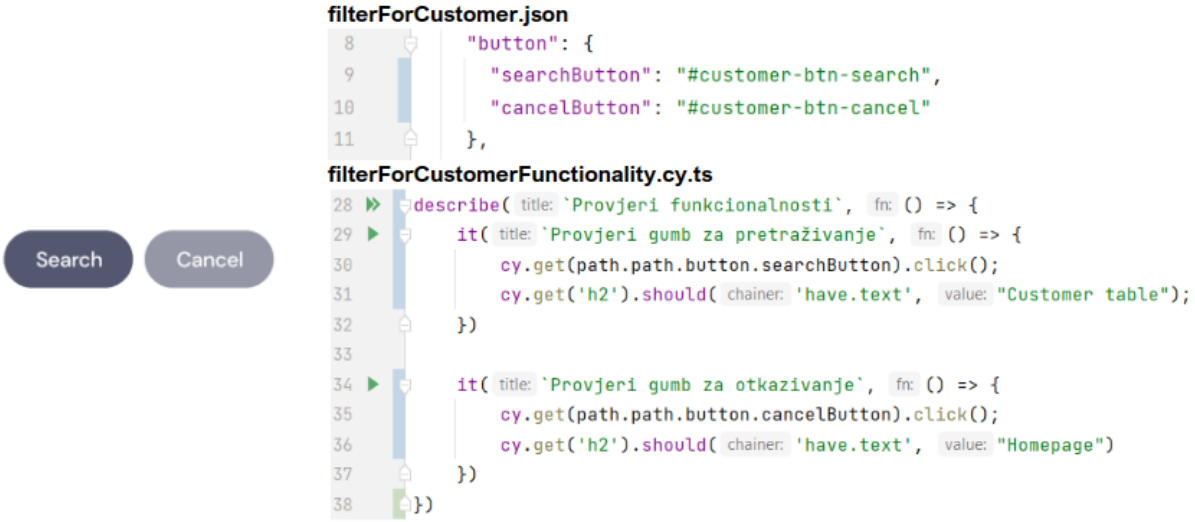
Slika 18. Testiranje hiperveze na kartici

U odabranom slučaju, otvorena je stranica za filtriranje kupaca. Filteri su podijeljeni u tri kartice, ukoliko je filter odabran u jednoj od kartica kraj nje se pojavljuje točka koja označuje da je unutar iste odabran filter te su u podnožju ispisani svi odabrani filteri u obliku znački. Prikazano je testiranje funkcionalnosti filtriranja te postojanost značke filtera nakon što je isti odabran. Test je raspisan u "filterForCustomerFunctionality.cy.ts" datoteci. Prije testa definirana je konstanta "allBadges" koja dohvaća sve string-ove iz .json datoteke. Za svaku značku koju dohvati putanjom iz "filterForCustomer.json" datoteke, stvara se novi test koji provjerava njezinu postojanost na stranici. Unutar naziva testa dinamički se dohvaća naziv značke pomoću "${badge}" varijable. U testu, za svaki "select" filter u prvoj kartici provjerava se postoji li značka nakon što se odabere funkcijom "checkSelectBadgeContent" iz "buzzOverview.ts" datoteke. Funkcija dohvaća .json putanju i unutar padajućeg izbornika za "select" element odabire prvu vrijednost te se nakon toga provjerava postoji li značka s tom vrijednošću dohvaćanjem "span.label" elementa i provjerom njegove vidljivosti. (Slika 19.)



Slika 19. Testiranje filtriranja podataka

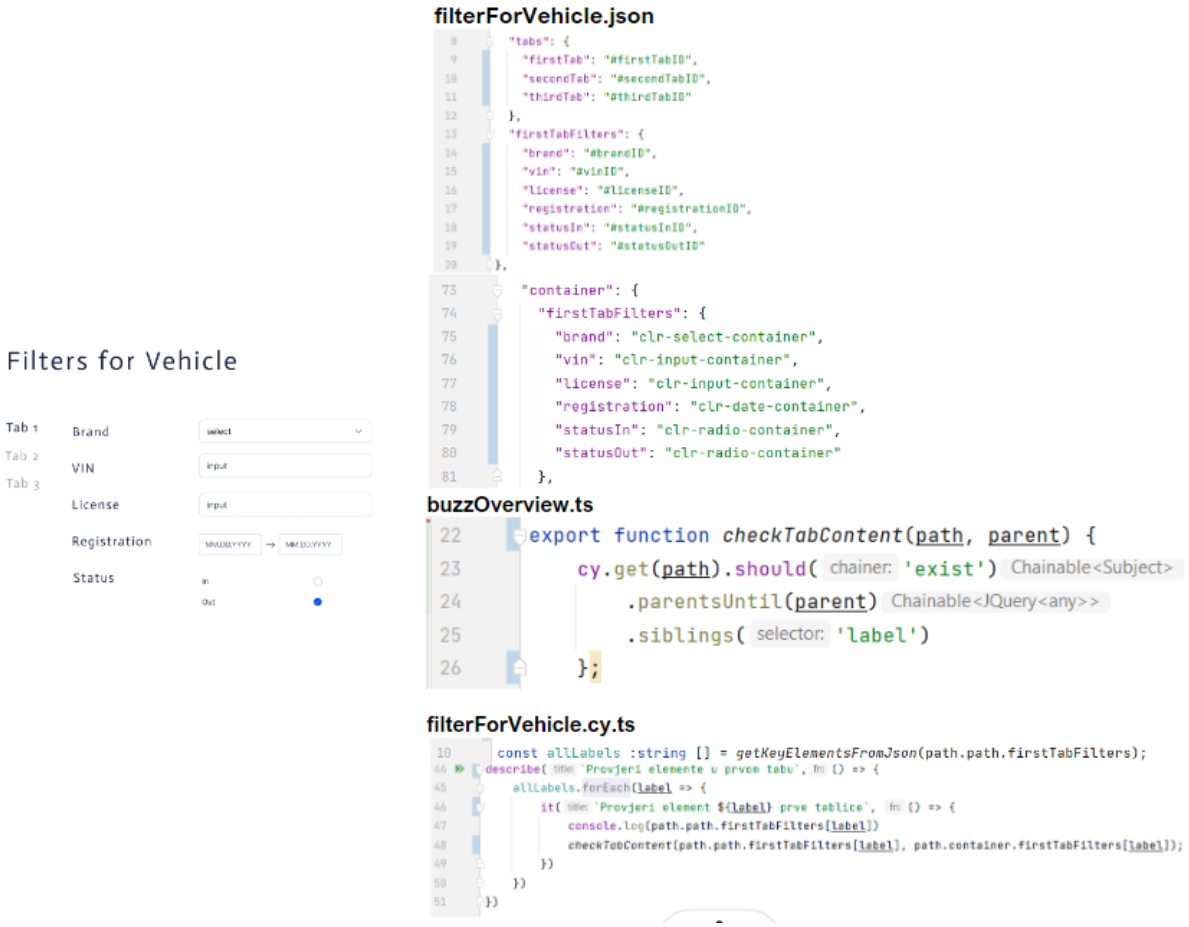
Na stranici se nalaze dva gumba gdje "search" vodi na stranicu s filtriranim rezultatima, a "cancel" poništava sve radnje i vraća natrag na naslovnu stranicu. Ovakve funkcionalnosti nalaze se i na stranici za filtriranje vozila uz dodatne ili promijenjene filtere. Iz "filterForCustomer.json" datoteke dohvaća putanju na gumb za pretraživanje, zatim klikom na gumb traži "h2" element koji sadrži tekst "Customer table". Ukoliko je element sa definiranim tekstom pronađen, prvi test prolazi. Drugi test provjerava funkcionalnost gumba za otkazivanje radnje s koracima kao i u prethodnom testu, ali ovaj put ide na početnu stranicu i "h2" element sadrži drugačiji naslov. (Slika 20.)



Slika 20. Testiranje gumba za pretraživanje i gumba za otkazivanje radnje

Isto tako postoji stranica za filtriranje podataka o vozilu. U ovom primjeru, testira se postojanost elemenata u prvoj kartici, a isti testovi izrađuju se za preostale dvije kartice. Prije

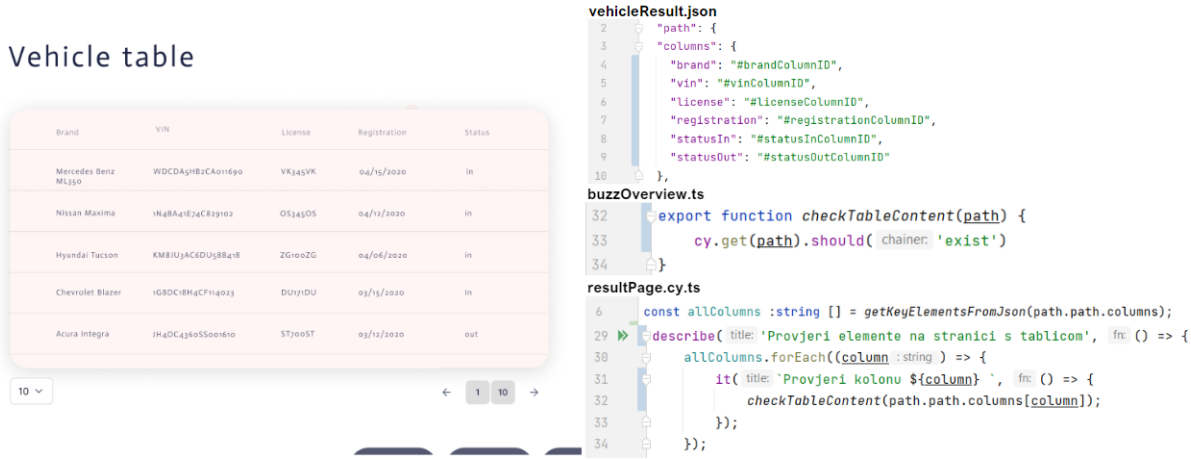
testa definirana je konstanta “allLabels” koja dohvaća sve string-ove iz .json datoteke. Za svaki element koju dohvati putanjom iz “filterForVehicle.json” datoteke, stvara se novi test koji provjerava postojanost istog elementa na stranici. Unutar naziva testa dinamički se dohvaća naziv značke pomoću “${label}” varijable. U testu, za svaki element u prvoj kartici provjerava se postoji li element koji se odabere funkcijom “checkTabContent” iz “buzzOverview.ts” datoteke. (Slika 21.)



Slika 21. Testiranje postoje li elementi za filtriranje podataka o vozilu unutar prve kartice

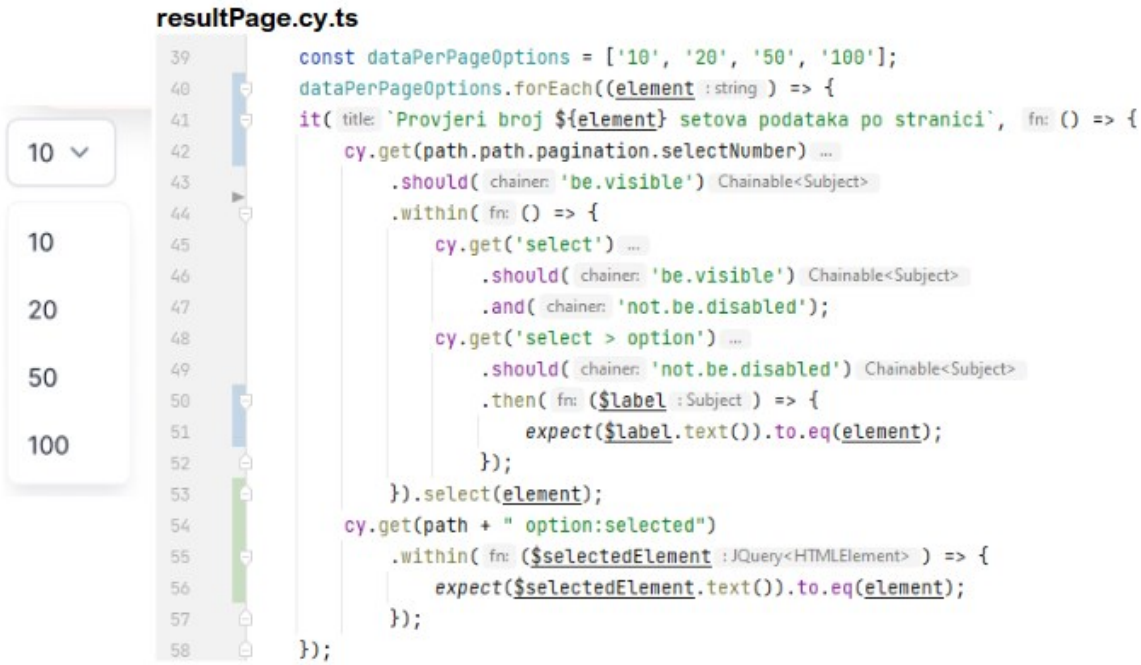
Nakon što se napravi pretraga prema odabranim filterima ili bez filtera, otvara se stranica s rezultatima prikazanim unutar tablice. U podnožju tablice s lijeve strane nalazi se padajući izbornik unutar kojeg se odabire broj setova podataka u tablici po stranici, a s desne strane nalazi se paginator koji omogućava kretanje kroz stranice tablice. Test je raspisan u “resultPage.cy.ts” datoteci. Prije testa definirana je konstanta “allColumns” koja dohvaća sve string-ove iz .json datoteke. Za svaku kolonu koju dohvati putanjom iz .json datoteke, stvara se novi test koji provjerava njezinu postojanost na stranici. Unutar naziva testa dinamički se

dohvaća naziv kolone pomoću "${column}" varijable. Korištenjem funkcije "checkTableContent" iz "buzzOverview.ts" datoteke, provjerava se postoji li kolona koju dohvaća iz "vehicleResult.json" dokumenta putanjom. (Slika 22.)



Slika 22. Testiranje elemenata na stranici na kojoj se prikazuju rezultati unutar tablice

Korisnik može odabrati broj setova podataka unutar tablice koristeći se padajućim izbornikom. Prikazan je test koji testira upravo tu funkcionalnost. (Slika 23.) Budući da se brojevi setova podataka ne mijenjaju, oni su unaprijed definirani unutar konstante "dataPerPageOptions". Prvi korak u testu je dohvatiti "select" element koji treba biti vidljiv, zatim provjeriti sve vrijednosti unutar padajućeg izbornika da odgovaraju definiranim brojevima koji se u testu pozivaju putem "element" varijable te da se odabere jedan element. Odabrani broj treba odgovarati definiranom broju. Unutar naziva testa dinamički se dohvaća broj setova podataka pomoću "${element}" varijable.



Slika 23. Testiranje funkcionalnosti postavljanja broja stranica unutar tablice

Pri dnu stranice s desne strane, nalaze se tri gumba. Gumb “Save” omogućava spremanje izvještaja unutar podizbornika. Nastavno s tim, “Load” gumb otvara podizbornik na istoj stranici u kojem se nalaze spremljeni i dijeljeni izvještaji. Gumb “New” vodi na naslovnu stranicu gdje je moguće započeti ponovno kreiranje izvještaja. Spremanjem izvještaja klikom na gumb za spremanje, otvara se skočni prozor gdje se upisuje naziv izvještaja te se klikom na gumb za spremanje unutar skočnog prozora isti izvještaj sprema. Međutim, moguće je prekinuti radnju spremanja klikom na gumb za otkazivanje radnje. Prikazan je test spremanja jednog izvještaja. (Slika 24.) Koraci u testu su sljedeći:

1. dohvati gumb spremanje izvještaja na stranici putanjom iz “customerResult.json” datoteke i klikni na njega
2. dohvati “h3” element koji predstavlja naslov u skočnom prozoru i uvjeri se da je naslov “Save the table”
3. dohvati element za unos teksta
4. unesi naziv “Automatizirani test”
5. dohvati gumb za spremanje u skočnom prozoru i klikni na taj gumb
6. dohvati gumb za učitavanje podizbornika na stranici i klikni na taj gumb

7. dohvati naslov kartice spremljenog izvještaja u podizborniku i uvjeri se da spremljeni izvještaj ima naslov “Automatizirani test”.

Posljednji dio testa uključuje i brisanje istog izvještaja kako se bespotrebno ne bi nakupljali spremljeni izvještaji i time usporavali rad aplikacije prilikom izvođenja testova. Koraci u brisanju su:

1. dohvati unutar kartice spremljenog izvještaja gumb padajućeg izbornika i klikni na njega
2. pronađi gumb za brisanje izvještaja i klikni na njega
3. u skočnom prozoru pronađi poruku upozorenja “Delete”
4. dohvati gumb za brisanje i klikni na njega.

Slika 24. Testiranje funkcionalnosti spremanja izvještaja

Skočni prozor za spremanje izvještaja ima još nekoliko funkcionalnosti. Jedna od njih je otkazivanje spremanja klikom na gumb za otkazivanje. Klikom na gumb za otkazivanje, skočni prozor se zatvara i radnja se prekida. Prikazan je test koji dohvaća gumb za spremanje izvještaja koji otvara skočni prozor te se uvjerava radi li se o skočnom prozoru za spremanje izvještaja te nakon toga klikom na gumb za otkazivanje spremanja uvjerava se da ne postoji naslov iz skočnog prozora na stranici. (Slika 25.)

Slika 25. Testiranje funkcionalnosti otkazivanja radnje spremanja izvještaja

Sljedeće ponašanje skočnog prozora za spremanje je upozorenje o nedovoljnom broju znakova. Test na sljedećem prikazu prikazuje test koji istim koracima kao i u prethodnim testovima otvara skočni prozor, no ne upisuje nikakav naziv za izvještaj te klikom na gumb za spremanje provjerava pojavljuje li se poruka upozorenja koja navodi kako je potrebno najmanje 2, a najviše 100 znakova za spremanje. Isti test se kreira za provjeru poruke upozorenja nakon što je upisano više od 100 znakova dodavanjem dodatnog koraka gdje se upisuje naziv od 101 znak. (Slika 26.)

Slika 26. Testiranje poruke upozorenja za spremanje izvještaja bez naziva

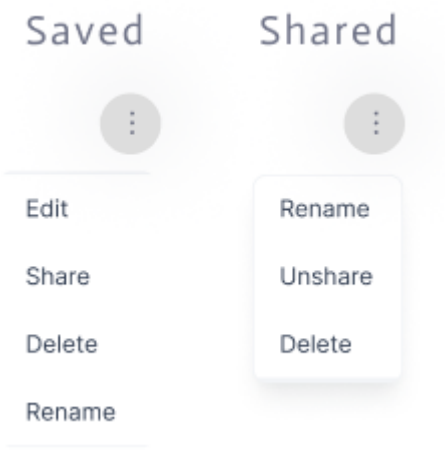
Posljednje ponašanje unutar skočnog prozora uključuje poruku upozorenja za spremanje izvještaja pod već postojećim nazivom. Iako slika prikazuje jedan test, on je unutar podijeljen na tri dijela kako bi se jednostavnije prepoznale radnje. (Slika 27.) Prvi dio uključuje kreiranje izvještaja pod jednim nazivom, istim koracima kao i u prethodno objašnjenim testovima. Drugi dio uključuje spremanje izvještaja s istim nazivom kroz korake:

1. dohvati gumb za kreiranje potpuno novog izvještaja na stranici i klikni na njega
2. zatim dohvati “h2” element i uvjeri se da ima tekst “Homepage”
3. dohvati karticu kupca i klikni na nju
4. na otvorenoj stranici dodaj filtere istim koracima kao i u testu na Slici 18.
5. dohvati gumb za pretraživanje i klikni na njega
6. na otvorenoj stranici dohvati gumb za spremanje i klikni na njega
7. dohvati “h3” element u skočnom prozoru i uvjeri se da je tekst “Save”
8. dohvati element za unos teksta i unesi isti naziv kao i već kreirani izvještaj u prvom dijelu testa
9. dohvati gumb za spremanje u skočnom prozoru i klikni na njega
10. dohvati poruku upozorenja i uvjeri se da piše “Name already exists.”
11. dohvati gumb za otkazivanje radnji i klikni na njega.

Nakon toga, dolazi treći dio u kojem se izvršava brisanje kreiranog izvještaja jednako kao što je opisano u testu prikazanom na Slici 24.

Slika 27. Testiranje poruke upozorenja za izvještaj s istim imenom

Klikom na gumb “Load” učitava se podizbornik s desne strane tablice, podijeljen na dvije grupe: “spremljeno” i “podijeljeno”. Korisniku je na ovaj način omogućeno pristupanje izvještajima koje je spremio s određenim filterima. Kraj svakog izvještaja nalazi se ikona s tri točkice, klikom na ikonu otvaraju se dodatne mogućnosti u obliku padajućeg izbornika. Spremljeni izvještaji i podijeljeni izvještaji imaju različite mogućnosti unutar padajućeg izbornika. Spremljene izvještaje moguće je urediti, podijeliti, obrisati ili preimenovati. Dok se podijeljene izvještaje može preimenovati, prestati dijeliti ukoliko je korisnik vlasnik izvještaja ili obrisati. (Slika 28.)



Slika 28. Prikaz funkcionalnosti u padajućim izbornicima unutar spremljenih i dijeljenih izvještaja

Sljedećim testom testira se otvaranje podizbornika klikom na gumb za učitavanje istog. Iako su neki prethodni testovi u koracima imali otvaranje podizbornika, kao važan dio aplikacije izdvojen je kao zaseban. Prikazan je test koji otvaranjem podizbornika klikom na gumb za učitavanje dohvaća “h3” element koji treba biti vidljiv i sadržavati tekst “Saved”. (Slika 29.)

Slika 29. Testiranje gumba za učitavanje podizbornika

Klikom na jednu od tih mogućnost uvijek se otvara skočni prozor, prikazana su dva primjera, jedan za mogućnost dijeljenja izvještaja, a drugi za poništavanje dijeljenja izvještaja. Skočni prozori omogućavaju nastavak procesa ili njegov prekid. Test je vizualno podijeljen u 7 dijelova radi jednostavnijeg snalaženja, a većina koraka je do sada objašnjena kroz prethodne testove. (Slika 30.) Prvi dio testa uključuje utvrđivanje naslova stranice na kojoj se nalazi i odabir kartice na početnoj stranici. Drugi dio testa uključuje postavljanje filtera i pretraživanje izvještaja prema filterima. Treći dio testa uključuje spremanje izvještaja gumbom za spremanje i unošenjem naziva izvještaja u element za unos teksta. Kako bi se poboljšala kvaliteta testa, koristi se Math.random() metoda koja generira nasumično brojeve prilikom upisivanja naziva za izvještaj. Na taj način test neće padati ukoliko se pokrene više puta, a prekine na pola jer svaki novi kreirani izvještaj ima jedinstveni naziv. Na kraju trećeg dijela, dohvaća gumb za spremanje u skočnom prozoru i klikne na njega. Četvrti dio testa otvara podizbornik i dohvaća naslov kartice izvještaja i provjerava sadrži li onaj tekst koji je upisan prilikom spremanja. Peti dio je važan dio testa, a koraci su sljedeći:

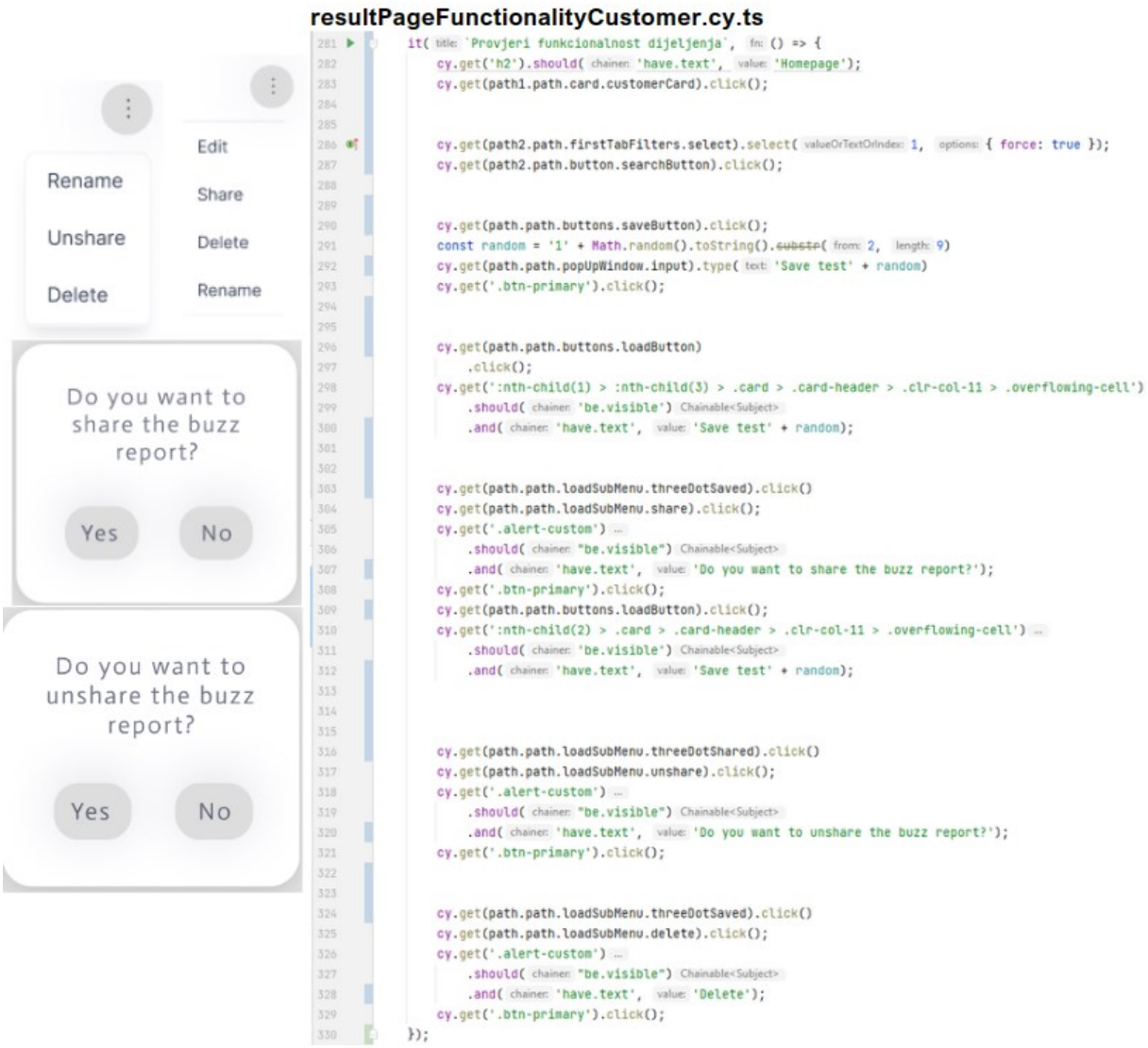
1. unutar podizbornika u dijelu spremljenih izvještaja pronađi padajući izbornik i klikni na njega
2. unutar padajućeg izbornika pronađi gumb za dijeljenje i klikni na njega
3. provjeri otvara li se skočni prozor dohvaćanjem poruke koja treba sadržavati tekst “Do you want to share the buzz report?”
4. dohvati gumb za potvrđivanje radnje i klikni ga

5. ponovi radnju s otvaranjem podizbornika i pronađi izvještaj s nazivom “Save test + random” u dijelu dijeljenih izvještaja.

Šesti dio također je važan dio testa, a predstavlja poništavanje dijeljenja izvještaja. Koraci u šestom dijelu su:

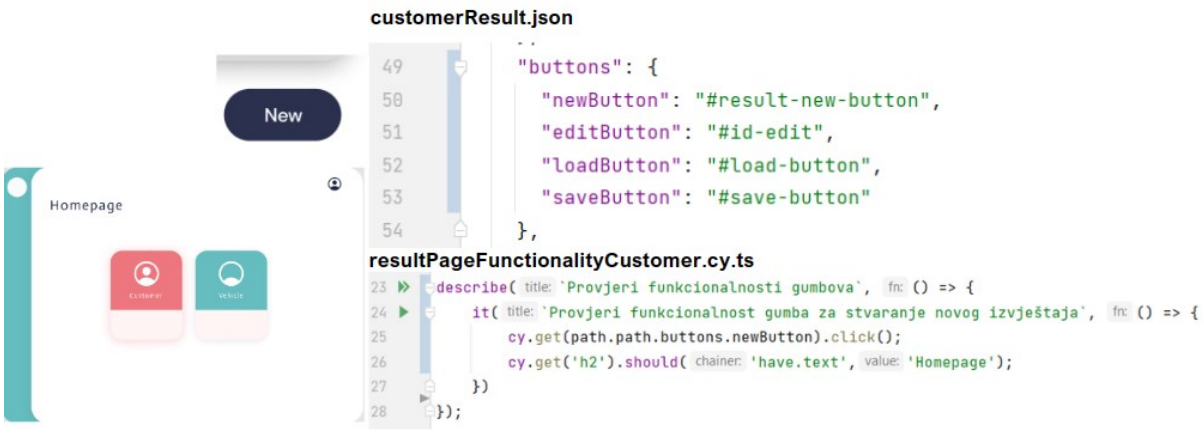
1. unutar podizbornika u dijelu dijeljenih izvještaja pronađi padajući izbornik i klikni na njega
2. unutar padajućeg izbornika pronađi gumb za poništavanje dijeljenja i klikni na njega
3. provjeri otvara li se skočni prozor dohvaćanjem poruke koja treba sadržavati tekst “Do you want to unshare the buzz report?”
4. dohvati gumb za potvrđivanje radnje i klikni ga.

Sedmi dio sadrži korake za brisanje kreiranog izvještaja kao i prethodni testovi.



Slika 30. Testiranje funkcionalnosti dijeljenja izvještaja

Nakon što je korisnik kreirao izvještaj i podijelio isti s drugim korisnicima, započinje sve ispočetka kreiranjem novog izvještaja vraćanjem na početnu stranicu klikom na gumb za stvaranje novog izvještaja. Upravo posljednji test testira gumb za kreiranje novog izvještaja. Iako se koristio u prethodnim testovima, također predstavlja važan dio aplikacije te je kao takav izdvojen. Dohvaćanjem gumba za kreiranje novog izvještaja i klikom na njega, na novootvorenoj stranici dohvaća “h2” element i uvjerava se sadrži li naslov tekst “Homepage”. (Slika 31.)



Slika 31. Testiranje funkcionalnosti gumba za stvaranje novog izvještaja

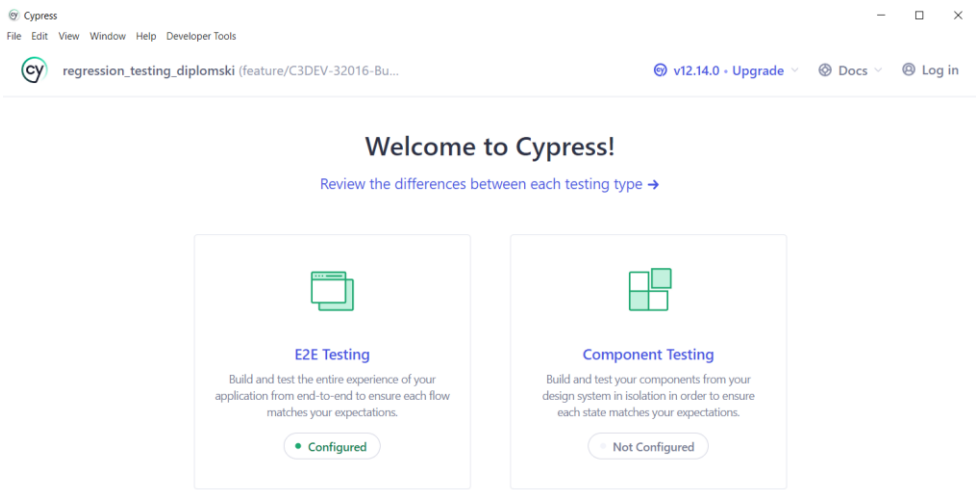
Naposljetku, sve napisane testove potrebno je izvršiti. Unutar IntelliJ IDEA kao i ostalim integriranim razvojnim okruženjima, Cypress testovi se izvršavaju naredbama u terminalu. Cypress će prema zadanim postavkama, koristeći se "npx Cypress run" naredbom, bezglavo (engl. *headless*) izvršavati testove, odnosno unutar terminala bez sučelja. (Slika 32.)

```
PS C:\ confidential \regression_testing_diplomski> npx cypress run
  Browser:        Electron 106 (headless)
  Node Version:   v18.9.0 (C:\Program Files\nodejs\node.exe)
  Specs:          7 found (startPage.cy.ts, customer/resultPage.cy.ts, vehicle/resultPage.cy.ts,
                   customer/functionality/filterForCustomerFunctionality.cy.ts, customer/functio
                  nality/resultPageFunctionalityCustomer.cy.ts, vehicle/functionality/filterForV
                  ehicleFunctionality.cy...)
  Searched:       cypress/e2e/**/*.cy.{js,jsx,ts,tsx}


  Running:  startPage.cy.ts                                                    (1 of 7)
```

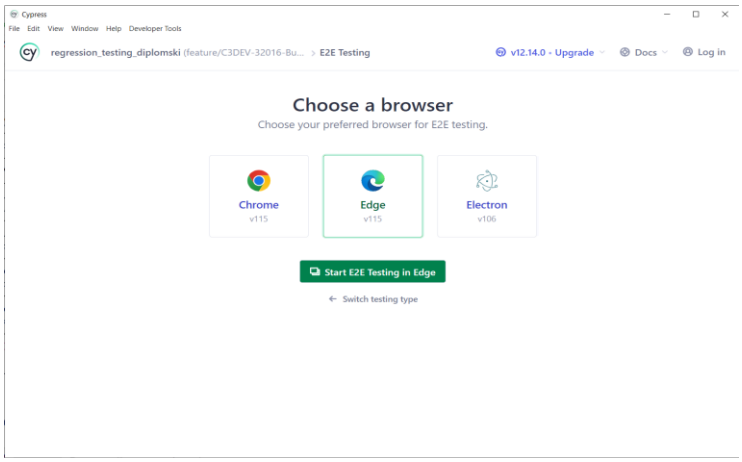
Slika 32. Izvršavanje testova naredbom "npx cypress run"

Dodavanjem navedenoj naredbi nastavak "--headed" (mora biti postavljen kad se vrti u pipline na gitlabu) prisilit će prikazivanje preglednika ili korištenjem naredbe "npx Cypress open" otvara se njegovo korisničko sučelje. (Slika 33.) Nastavak "--headed" važno je postaviti kada se testovi vrte u CI/CD pipeline na Gitlabu.



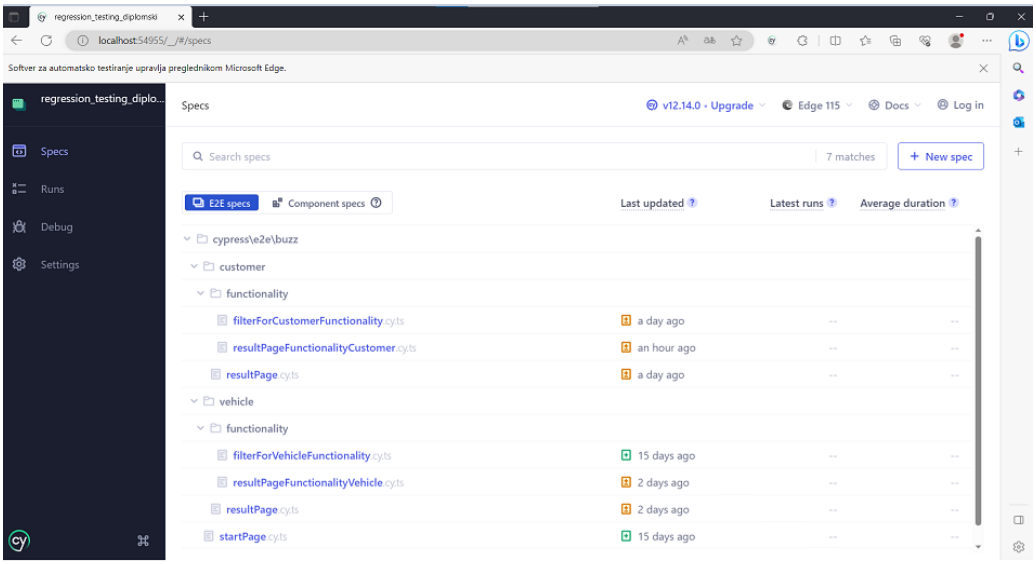
Slika 33. Cypress sučelje otvoreno izvršavanjem naredbe “npx cypress open”

Na početnom zaslonu odabire se tip testova koji se žele izvršiti, odabirom “E2E Testing” tipa otvara se nova stranici na kojoj se odabire mrežni preglednik unutar kojeg će se izvršavati testovi. (Slika 34.)



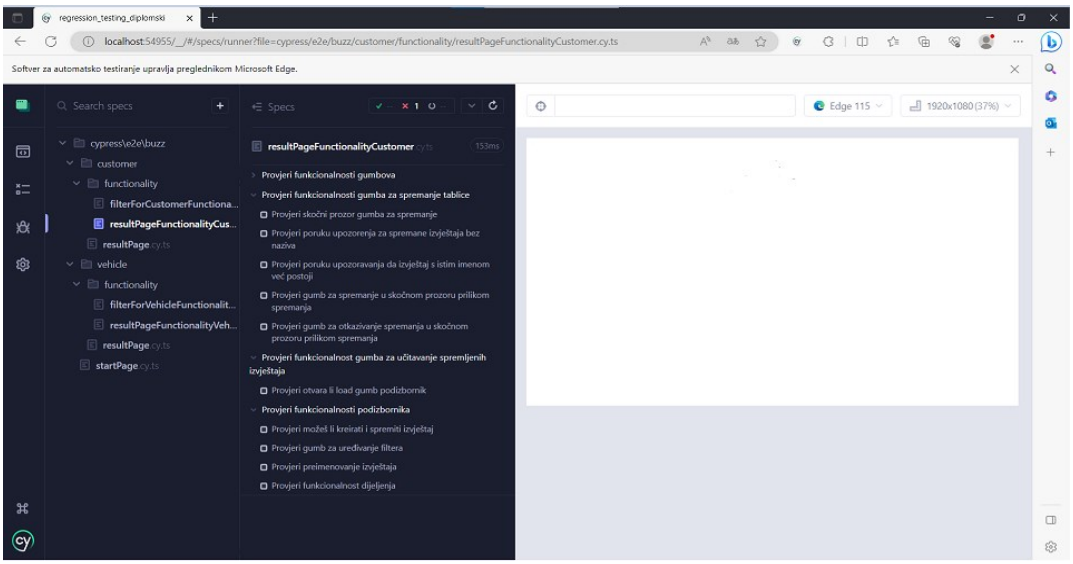
Slika 34. Cypress sučelje gdje se odabire mrežni preglednik

Nakon odabranog mrežnog preglednika, otvara se projekt. (Slika 35.) Svaki testna datoteka otvara se klikom.



Slika 35. Cypress sučelje gdje je prikazan projekt

Klikom na jednu testnu datoteku, npr. “resultPageFunctionalityCustomer.cy.ts” otvaraju se svi napisani testovi unutar te datoteke koji se automatski izvršavaju. (Slika 36.)



Slika 36. Cypress sučelje gdje se izvode testovi

Nakon svih izvršenih testova, dobivanje izvještaja o rezultatima objašnjeno je u trećem poglavlju. Tim posljednjim korakom, završena je implementacija automatiziranih testova, a slijedi analiza izvještaja, potencijalno prijavljivanje bugova i nastavak ručnog dijela testiranja.

5. Izazovi

Najveći izazov prilikom stvaranja automatiziranih testova je organizacija istih, iščitavanje dokumentacije za osmišljavanje najefikasnijeg rješenja kako bi testovi što duže ostali dosljedni te kreirani prema dobroj praksi. Iz takvih razloga, odlučeno je koristiti .json datoteke. Kao što je već navedeno u radu, takvi dokumenti omogućuju brže i jednostavnije promjene ukoliko dođe do promjene u elementu. Na primjer, u .json datoteci popisani su svi ID-jevi "button" elemenata koji se nalaze na početnoj stranici te se naknadno odluči dodati nova funkcionalnost, ali se promjene i ostali ID-jevi. U takvom slučaju, dovoljno je na jednom mjestu promijeniti ID-jeve i dodati jedan dodatni, nego u svakom testu gdje se koristi ići zasebno mijenjati ID. Isto tako je važno podijeliti testiranje funkcionalnosti od ostalih testova i razmišljati proaktivno što bi se novo moglo pojaviti kako bi u budućnosti mogli bolje organizirati. Ovaj izazov, općeniti je izazov pri svakom novom zadatku, stoga su sljedeći usko vezani uz same testove.

Iako je asinkronost posebnost Cypress testova, mnogi koji započinju s korištenjem Cypress-a nakon primjerice Seleniuma i Jave naići će na neke prepreke. Stoga je bilo važno shvatiti kako Cypress naredbe u pozadini odrađuju dosta radnji koje bi inače pisali kao korake u testovima u Selenium-u. Najjednostavniji primjer toga je "await" koji se uopće ne koristi u Cypress testovima, dok se "wait" često piše u Selenium testovima. Naposljetku, onaj koji uči pisanje automatiziranih testova u Cypress-u i JaviScript/TypeScript programskom jeziku, jednostavnije prihvaća asinkronost Cypress-a.

Još jedan od izazova unutar Cypress-a predstavljao je "iframe". Iako se izgled početne stranice promijenio, a zadani preglednik postao Edge te rješenje koje se pronašlo više nije bilo potrebno jer "iframe" više nije bio dio koda. Ipak, dio je ovog procesa te će se kao takav prikazati unutar ovog poglavlja. Dohvatiti gumb unutar "iframe" elementa nije bilo moguće, test bi svaki put pucao, koliko god naredbi se postavilo da dohvati točan dio koji treba kliknuti. Nakon istraživanja, rješenje je bilo jednostavno. U "cypress.json" datoteku je bilo potrebno postaviti svojstvo "{"chromeWebSecurity": false}" koji omogućuje pristup "iframe" elementu s više domena. Razlog tomu je činjenica da se sve ugrađene naredbe za obilaženje DOM-a teško zaustavljaju u trenutku kada naiđu na "#document" unutar "iframe" elementa. Stoga je bilo važno prilagoditi kod, a zahvaljujući svojstvu u "cypress.json" dokumentu, testovi su prošli.[32]

[32] Usp. Bahmutov, Gleb. Working with iframes in Cypress, 2020. URL: https://www.cypress.io/blog/2020/02/12/working-with-iframes-in-cypress/ (2023-08-03)

Od ostalih izazova koji nisu bili vezani uz kod je pisanje testova za aplikaciju koja svakim sprintom ima nove funkcionalnosti za koje je potrebno kontinuirano i usporedno stvarati nove testove i dodavati elemente. Potrebno je dosta pažnje i vremena za kreiranje testova, usklađivanje verzija, praćenja novih ažuriranja i sl. To čini posao kreiranja automatiziranih testova beskrajnim, no u krajnjem slučaju dobrim na duge staze.

6. Zaključak

Već 70-ak godina programeri otklanjaju pogreške na svojim programima, a 40-ak godina smatraju testiranje kao dijelom životnog ciklusa. Pogreške su neizbježne u svakodnevnom radu, a pronalaženjem istih te njihovim otklanjanjem, stvara se kvaliteta koju upravo QA testeri održavaju. Započevši s ručnim testiranjem, dolazi se do automatiziranog testiranja. Automatizirano testiranje neizbježno je u razvoju jednog QA testera, kao logičan slijed u karijeri. Automatizirani testovi ostavljaju testeru vremena za ručno testiranje zahtjevnijih stvari za koje se još uvijek ne izrađuju automatizirani testovi, za dokumentaciju te učenje.

Proces automatiziranog testiranja programske podrške namijenjene autosalonima opisan je od samog početka postavljanja projekta. Prije pisanja automatiziranih testova, važno je imati dobro definirane zahtjeve sustava i dizajn rješenja iz kojih se mogu kreirati testovi za ručno testiranje. Tek nakon toga, moguće je napisati kvalitetne automatizirane testove koji u budućnosti neće zahtijevati velike promjene. Postavljanje projekta zahtjevan je proces kako bi se moglo snalaziti unutar direktorija tijekom kreiranja novih testova za nadolazeće funkcionalnosti. Uvijek je dobro pregledati postojeću dokumentaciju i pronaći najbolje prakse, a zahvaljujući velikoj Cypress zajednici, to je vrlo jednostavno.

U praktičnom dijelu rada prikazani su testovi usporedno s prikazom zaslona onoga što se testira. Svaka od stranica ima svoje funkcionalnosti za koje je bilo potrebno napisati automatizirane testove, sveukupno 5 stranica s pojedinim ponavljajućim funkcionalnostima, ali uglavnom posebnim za stranicu na kojoj se nalaze. Iako naizgled jednostavni, testovi su kreirani s puno razmišljanja o mogućim problemima koji bi se pojavili. Svakako ih je puno više nego što se jednim radom može prikazati, ali svi testovi sadrže djelomično iste korake pa se unutar rada prikazao skup onih testova koji objedinjuju ono što svi imaju. Prikazani testovi čine jedno korisnikovo putovanje aplikacijom što je bio cilj praktičnog dijela. Cypress-ovo sučelje jednostavno je za koristiti, a mogućnost dobivanja snimki zaslona onih dijelova gdje je pronađen bug uvelike pojednostavljuje proces automatizacije. Svi izazovi na koje se naišlo tijekom procesa dio su procesa učenja i unatoč izgubljenom vremenu na istima, dugoročno je uštađeno vrijeme. S obzirom da su testovi prvi puta pisani u Cypress okruženju i TypeScript programskim jezikom, bilo je potrebno određeno vrijeme kako bi se shvatila logika i način raspisivanja testova, kako bi se pritom pojednostavile stvari funkcijama. Automatizirano testiranje sadašnjost i budućnost je QA struke, a svojim postojanjem neće zamijeniti čovjeka koji ipak treba iste i napisati.

Literatura

1. Atlassian. URL: https://marketplace.atlassian.com/apps/1211769/xray-test-management-for-jira?tab=overview&hosting=cloud (2023-07-25)
2. Atlassian. URL: https://www.atlassian.com/agile/kanban/kanban-vs-scrum (2023-07-25)
3. Atlassian. URL: https://www.atlassian.com/software/jira?tab=plan (2023-07-25)
4. Atlassian. URL: https://www.atlassian.com/software/jira/features/kanban-boards (2023-07-25)
5. Atlassian. URL: https://www.atlassian.com/software/jira/features/scrum-boards (2023-07-25)
6. Atlassian. URL: https://www.atlassian.com/software/jira/guides/more/jira-family#what-is-the-jira-family (2023-07-25)
7. Automated regression testing: a detailed guide, 2023. URL: https://www.browserstack.com/guide/automated-regression-testing (2023-07-07)
8. Bahmutov, Gleb. Working with iframes in Cypress, 2020. URL: https://www.cypress.io/blog/2020/02/12/working-with-iframes-in-cypress/ (2023-08-03)
9. Chako, Oleksandr. Playwright vs Cypress: which framework to choose for E2E testing?. URL: https://eleks.com/research/playwright-vs-cypress/ (2023-07-24)
10. Cypress. URL: https://docs.cypress.io/guides/core-concepts/introduction-to-cypress (2023-07-25)
11. Cypress. URL: https://docs.cypress.io/guides/overview/why-cypress (2023-07-25)
12. Cypress. URL: https://docs.cypress.io/guides/tooling/reporters (2023-07-25)
13. Cypress. URL: https://www.cypress.io/about-us/our-story/ (2023-07-25)
14. Dustin, Elfriede; Garrett, Thom; Gauf, Bernie. Implementing automated software testing. Boston: Addison-Wesley, 2009.
15. Graham, Dorothy…[et. al.]. Foundations of software testing: ISTQB Certification. Australia…[et.al]: Cengage Learning Emea, 2008.

16. Gray box testing: software testing. URL: https://www.geeksforgeeks.org/gray-box-testing-software-testing/ (2023-07-05)

17. IBM. URL: https://www.ibm.com/topics/software-testing (2023-07-03)

18. Katalon. URL: https://katalon.com/resources-center/blog/end-to-end-e2e-testing-tools-frameworks (2023-07-24)

19. QA testing: what is DEV, SIT, UAT and PROD?, 2020. URL: https://medium.com/@buttertechn/qa-testing-what-is-dev-sit-uat-prod-ac97965ce4f (2023-07-07)

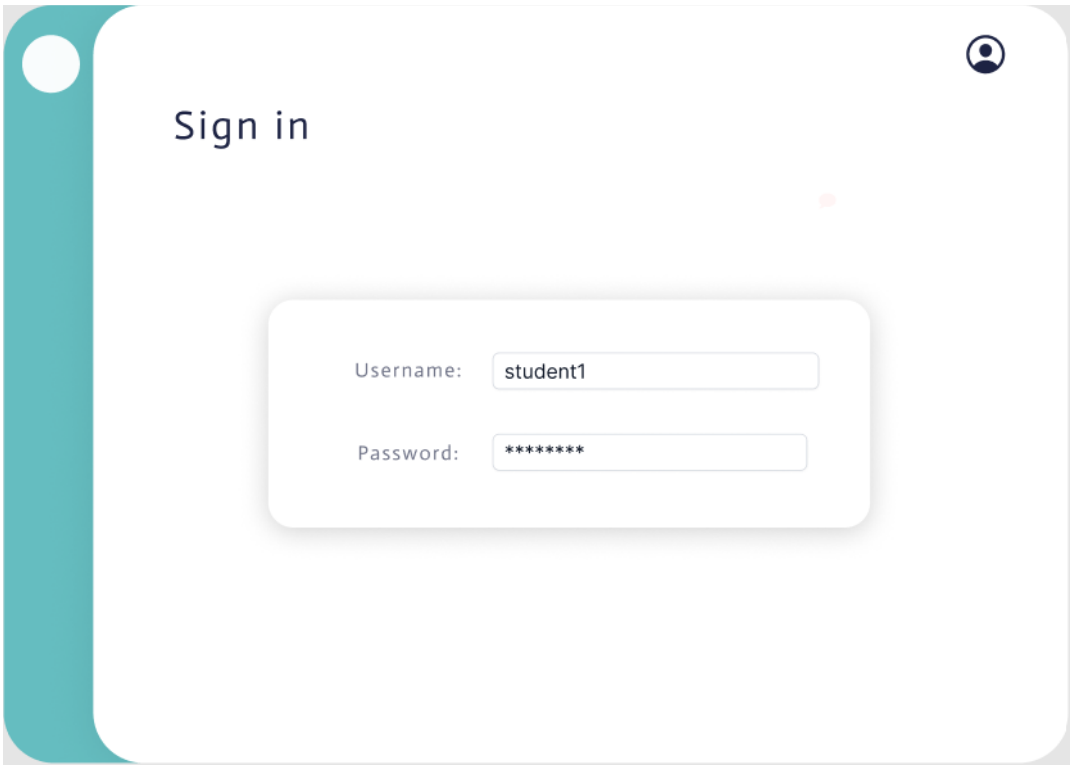
20. Safonova, Elena.Understanding the core positions in QA team and ways to hire them, 2022. URL: https://sumatosoft.com/blog/understanding-the-core-positions-in-qa-team-and-ways-to-hire-them (2023-07-03)

21. Software Testing Fundamentals. URL: https://softwaretestingfundamentals.com/ (2023-07-03)

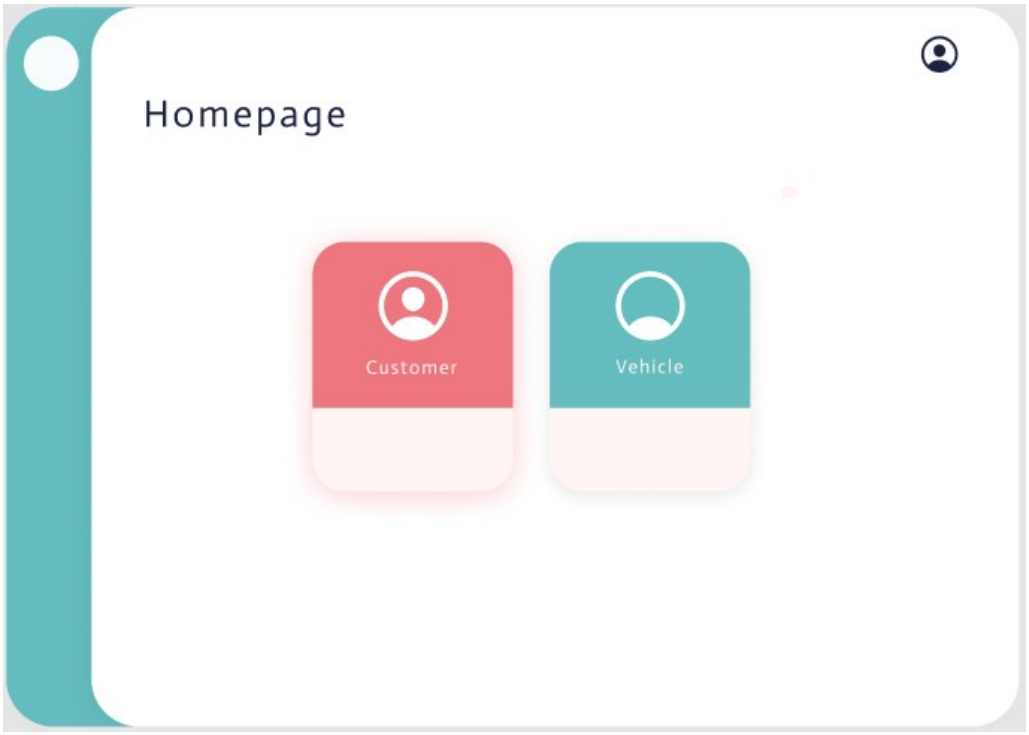
22. Whittaker, James A. Exploratory software testing. Boston: Addison-Wesley, 2009.

Prilozi

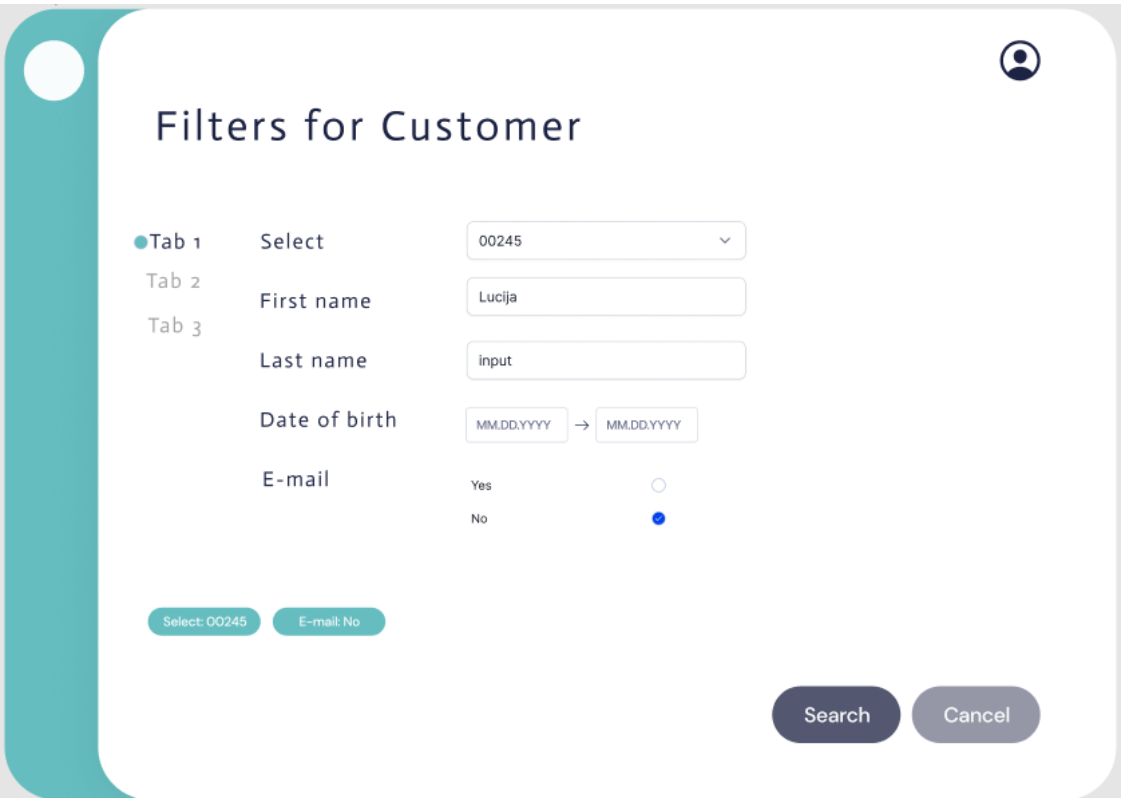
Snimke zaslona mrežne aplikacije.



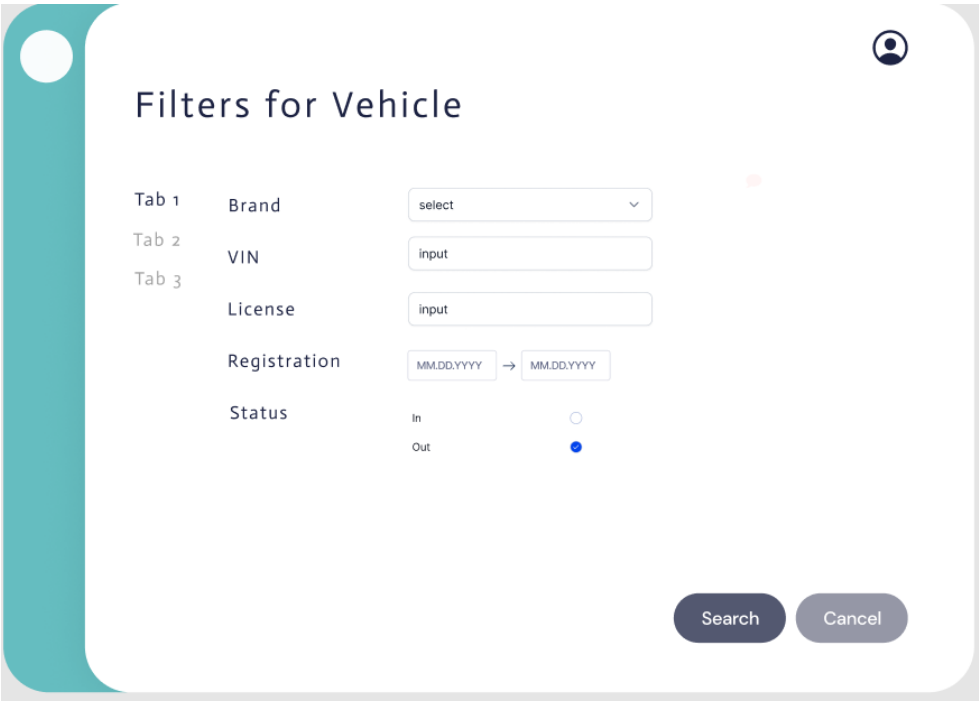
Slika 1. Snimka zaslona korisničkog sučelja za prijavu u mrežnu aplikaciju



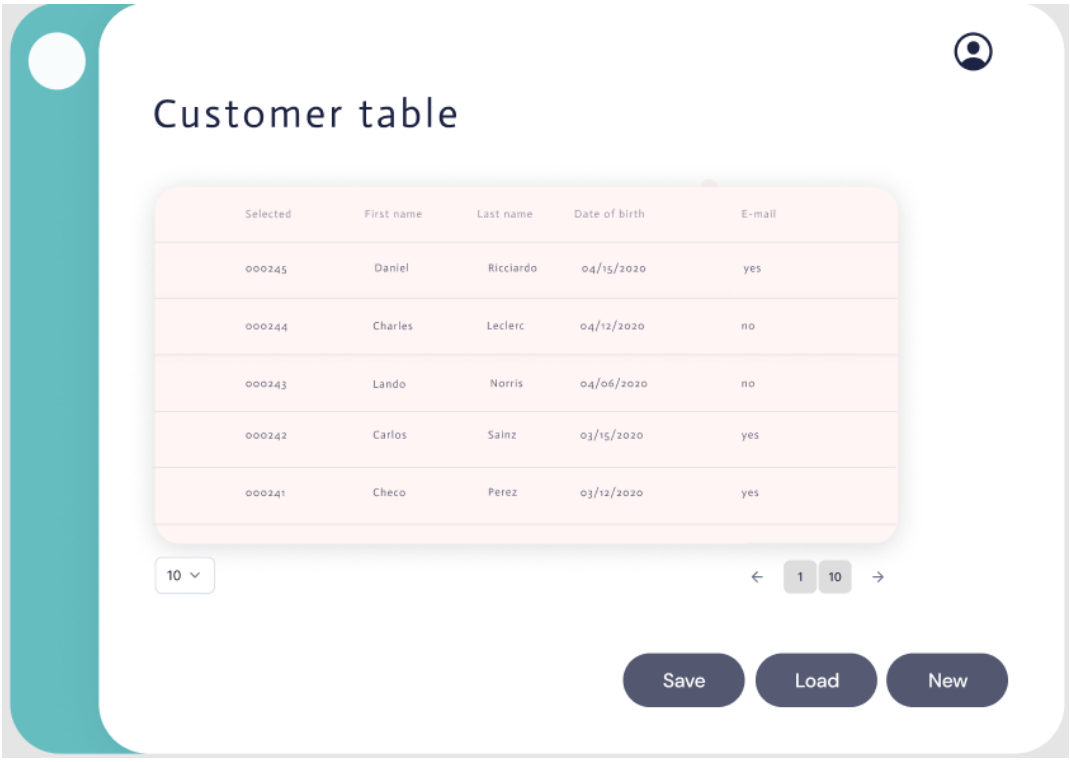
Slika 2. Snimka zaslona korisničkog sučelja početne stranice mrežnu aplikaciju



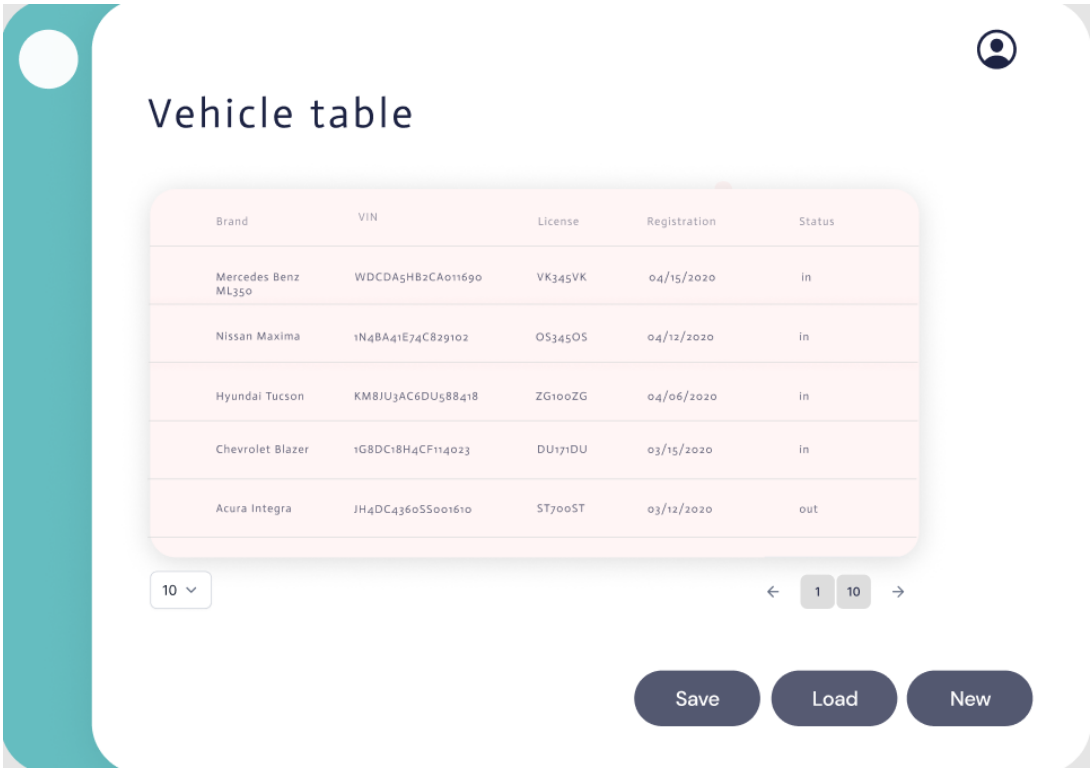
Slika 3. Snimka zaslona korisničkog sučelja dijela stranice za filtriranje podataka



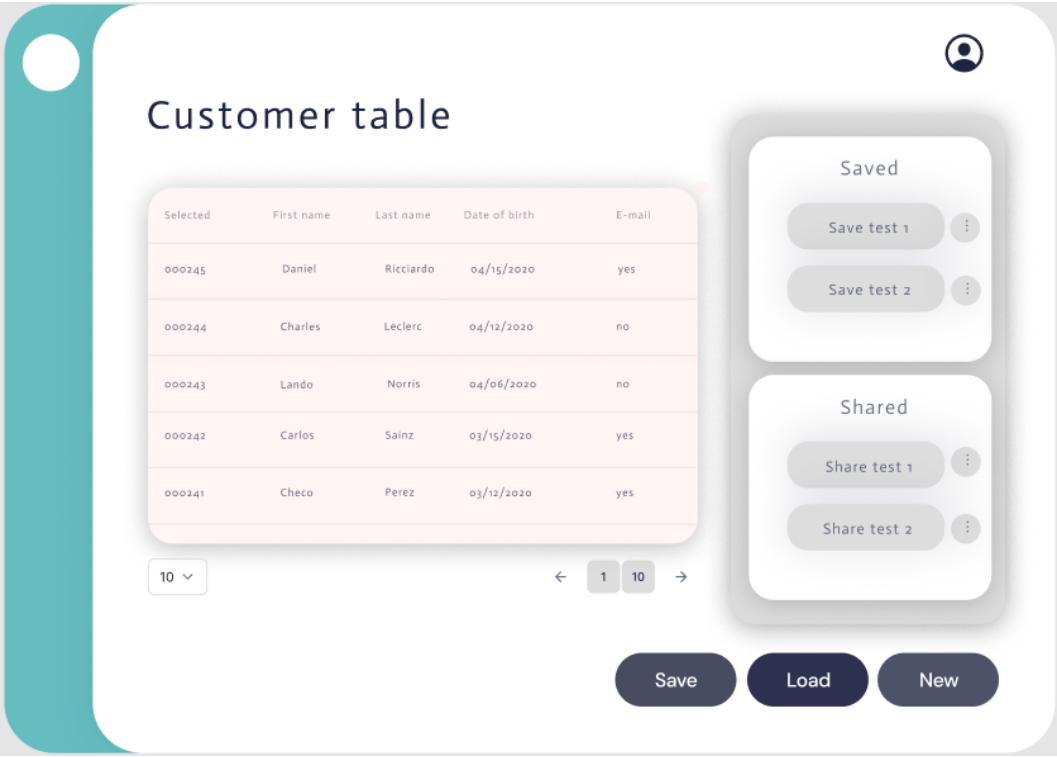
Slika 4. Snimka zaslona korisničkog sučelja dijela stranice za filtriranje podataka



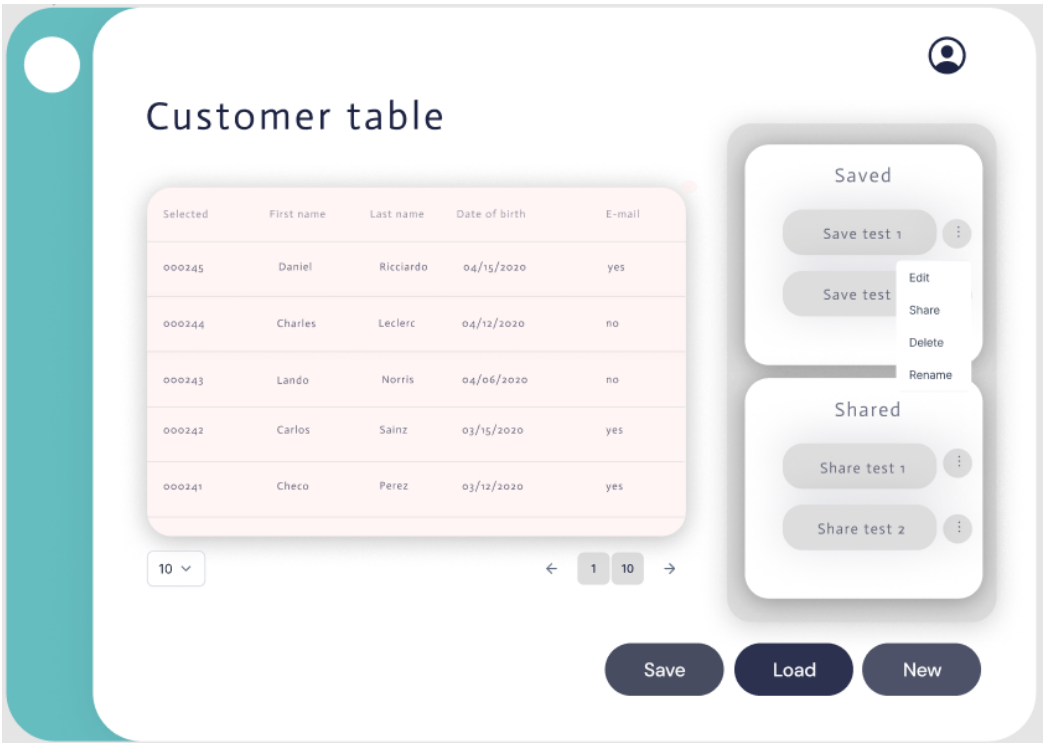
Slika 5. Snimka zaslona korisničkog sučelja dijela stranice s tablicom za kupce



Slika 6. Snimka zaslona korisničkog sučelja dijela stranice s tablicom za vozila



Slika 7. Snimka zaslona korisničkog sučelja dijela stranice s tablicom i otvorenim podizbornikom



Slika 8. Snimka zaslona korisničkog sučelja dijela stranice s tablicom i otvorenim padajućim izbornikom u podizborniku