

SVEUČILIŠTE JURJA DOBRILE U PULI  
FAKULTET INFORMATIKE

**Kristijan Cetina**

**Testiranje klijentskih komponenti web aplikacija pomoću alata Playwright**

DIPLOMSKI RAD

Pula, 2024.

SVEUČILIŠTE JURJA DOBRILE U PULI  
FAKULTET INFORMATIKE

**Kristijan Cetina**

**Testiranje klijentskih komponenti web aplikacija pomoću alata Playwright**

**DIPLOMSKI RAD**

<b>JMBAG:</b>	<b>2424011721, izvanredni student</b>
<b>Studijski smjer:</b>	<b>Informatika</b>
<b>Kolegij:</b>	<b>Diplomski rad</b>
<b>Znanstveno područje:</b>	<b>Društvene znanosti</b>
<b>Znanstveno polje:</b>	<b>Informacijske i komunikacijske znanosti</b>
<b>Znanstvena grana:</b>	<b>Informacijski sustavi i informatologija</b>
<b>Mentor:</b>	<b>dr.sc. Nikola Tanković</b>

Pula, travanj, 2024. godine

# Zahvala

Zahvaljujem svojem mentoru na izdvojenom vremenu i podršci, kako na izradi ovog rada, tako i tijekom cijelog studiranja na Fakultetu informatike.

Zahvaljujem se i svojim timskim kolegama, s kojima sam od samog početka sudjelovao na svim timskim zadacima i njihovoj pomoći pri individualnom radu.

Zahvaljujem se i svim ostalim profesorima i djelatnicima na nesebičnoj potpori kada je god to bilo potrebno.

Naposljetku veliko hvala mojoj obitelji na potpori i razumijevanju tijekom mojeg ponovnog studiranja.

Cool quote

# Izjava o samostalnosti izrade završnog rada

Izjavljujem da sam završni rad na temu *Testiranje klijentskih komponenti web aplikacija pomoću alata Playwright* samostalno izradio uz pomoć mentora, koristeći navedenu stručnu literaturu i znanje stečeno tijekom studiranja. Završni rad pisan je u duhu hrvatskoga jezika.

Student: Kristijan Cetina

## Sažetak

Sažetak HR

## Ključne riječi

*Playwright, JavaScript, open-source,*

## Sommario

Sommario IT

## Parole chiave:

*Playwright, JavaScript, open-source,*

## Abstract

Abstract EN

## Keywords:

*Playwright, JavaScript, open-source,*

# Popis oznaka i kratica

Oznaka	Opis	Jedinica
$t$	vrijeme (sekunda)	$s$
$\theta$	temperatura (Celzijev stupanj)	$^{\circ}C$
$\nu$	brzina	$m/s$
$s$	udaljenost u metrima	$m$
$f$	frekvencija	$Hz$
$C$	kapacitet kondenzatora	$F$
	Veličina memorije	MB, 1MB = 1048576 bajtova

Kratica	Opis
GPS	Global Positioning System - Sustav globalnog pozicioniranja
SD	Secure Digital - format memorijske kartice
$\mu$ SD, microSD	mikro Secure Digital - kartica manjih fizičkih dimenzija
PWM	Pulse Width Modulation - Pulsno-širinska modulacija
IDE	Integrated Development Environment - Integrirano razvojno okruženje
GND	Točka nultog potencijala
SW	Software
HW	Hardware
NMEA	National Marine Electronics Association
UTC	Coordinated Universal Time - Standardno vrijeme
.csv	comma-separated values - vrijednosti odvojene zarezom
.md	Markdown datoteka

## Korišteni strani pojmovi

Pojam	Opis
Product owner	osoba zadužena za određivanje prioriteta zahtjeva

# Sadržaj

<b>Sažetak</b>	<b>IV</b>
Ključne riječi . . . . .	IV
<b>Sommario</b>	<b>IV</b>
Parole chiave: . . . . .	IV
<b>Abstract</b>	<b>IV</b>
Keywords: . . . . .	IV
<b>Popis oznaka i kratica</b>	<b>V</b>
<b>0 Uvod i opis zadatka</b>	<b>1</b>
0.1 Opis i definicija problema . . . . .	1
0.2 Cilj i svrha rada . . . . .	1
0.3 Hipoteza rada . . . . .	1
0.4 Metode rada . . . . .	1
0.5 Struktura rada . . . . .	2
<b>1 Uvod u testiranje programskog rješenje i osiguranje kvalitete</b>	<b>3</b>
1.1 Proces testiranja . . . . .	3
1.2 Ciljevi testiranja . . . . .	4
1.3 Uloga testera u timu . . . . .	5
<b>2 Playwright</b>	<b>6</b>
2.1 Opis i pregled paketa . . . . .	6
2.2 Instalacija . . . . .	7
2.3 Generiranje testova . . . . .	9
2.4 Pokretanje alata za generiranje testova . . . . .	10
2.5 Kontinuirana integracija - CI . . . . .	11
<b>3 Testiranje nakon nadogradnje na novu verziju</b>	<b>12</b>
3.1 Postojeći način testiranja . . . . .	12
3.2 Automatsko testiranje procesa nadogradnje . . . . .	13
<b>4 Implementacija u produkciji na stvarnom proizvodu</b>	<b>15</b>
4.1 Uvođenje i inicijalizacija . . . . .	15
4.2 Deklaracija varijabli . . . . .	16
4.3 Konfiguracija testova . . . . .	16
4.4 Glavni test . . . . .	16
4.5 Pomoćne funkcije . . . . .	18

<b>5 Zaključak</b>	<b>19</b>
<b>Literatura</b>	<b>20</b>
<b>Popis slika</b>	<b>20</b>



# Poglavlje 0

## Uvod i opis zadatka

Tema ovog rada proizašla je iz autorove želje za proučavanjem tematike te kao gorljivim poklonikom metode učenja kroz praktičan rad i primjenu stečenog znanja i iskustva na rješavanje realnog problema.

### 0.1 Opis i definicija problema

gdfg

### 0.2 Cilj i svrha rada

yeryy

### 0.3 Hipoteza rada

Hipoteza ovog rada je da promjenom primjerenih metoda testiranja programskog proizvoda može se značajno smanjati količina grešaka (*bugova*, *engl. bugs*) u finalnom proizvodu koji se isporučuje krajnjem korisniku te ostvariti uštede u resursima za njihovo ispravljanje.

### 0.4 Metode rada

Tijekom izrade ovoga rada korištene su različite znanstveno-istraživačke metode od kojih je svaka najprikladnija postavljenom izazovu, a one su:

- Istraživačka metoda - za stjecanje uvida u zadane okvire zadatka
- Metoda logičke analize i sinteze - za prikupljanje podataka iz literature
- Deskriptivna metoda - za izradu uvodnog i završnog dijela projektnog zadatka
- Eksperimentalna metoda - u potrazi za optimalnim rješenjima za zadani dio problema

## 0.5 Struktura rada

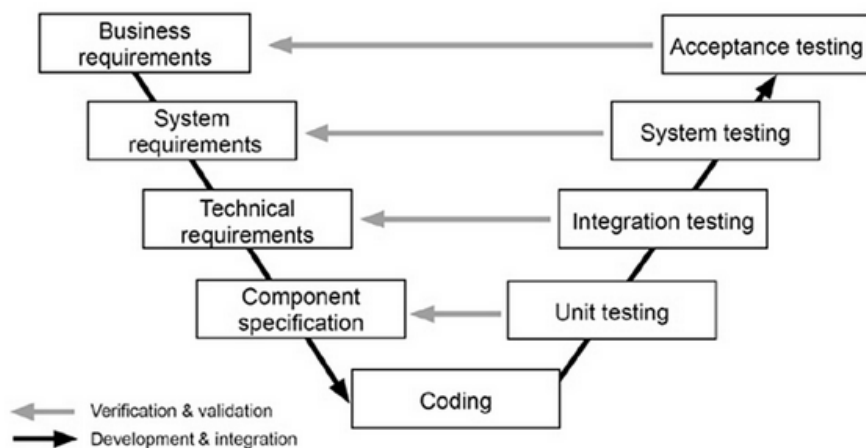
Struktura ovoga rada podjeljena je u logičke cjeline. Nakon uvoda i objašnjavanja rada, u poglavlju

Kompletan Git repozitorij ovog rada javno je dostupan na <https://github.com/KristijanCetina/jsTesting>

## Poglavlje 1

# Uvod u testiranje programskog rješenje i osiguranje kvalitete

Testiranje softwareskog proizvoda se provodi u cilju osiguravanja kvalitete samog proizvoda. Svaka faza razvoja ima svoje specifične zahtjeve i načine testiranja, a njihov pregled je dan na slici 1.1. Pojedine funkcije u kodu se pokrivaju unit testovima koji se brinu da pojedine komponente rade ono što su namjenjene na najnižoj razini. To su testovi koji se u pravilu izvršavaju vrlo često prilikom pisanja koda kako bi se osiguralo da promjena neke funkcije ili komponente nije poremetila njen rezultat.



Slika 1.1: Proces testiranja u Scrum metodologiji rada

### 1.1 Proces testiranja

Svaki proces testiranja započinje sa izradom plana testiranja unutar kojeg se definira šta se testira na koji način te koji su uvjeti da se zadani test smatra uspješnim (*engl. Acceptance criteria*).

Tokom samog razvijanja nove funkcionalnosti u softwareskom proizvodu često se izvršava ručno testiranje kako bi se utvrdilo da li proces razvoja ide u zadanom smjeru i u konačnici da se radi ono što je dogovoreno bilo interno s timom ili čak i sa klijentom.

Nakon što se završi sam razvoj nove funkcionalnosti onda se radi i završno funkcionalno testiranje i izrada automatskih testova koji će se u budućnosti izvršavati automatski u zadanom intervalu kako bi se osiguralo da uvođenje nove funkcionalnosti ne uvode nove pogreške na već ispravnim funkcionalnostima. Poznato kao i regresijsko testiranje.

Bitno je testirati realne scenarije koji se očekuju da moraju zadovoljiti, kao i one scenarije od kojih se očekuje da nesmiju proći test. To se radi u svrhu potvrde da test zaista radi ono što je namijenjen, a ne da imamo propust u samom testu koji uvijek vraća pozitivan rezultat ili da testirana funkcija nema implementiranu validaciju ulaznih parametara koji onda mogu izazvati nepoželjno ponašanje programa.

## 1.2 Ciljevi testiranja

Ciljevi testiranja moraju zadovoljavati nekoliko kriterija, a to su:

- Specifičnost
- Mjerljivost
- Ostvarljiv
- Realističan
- Vremenski ograničen

Neki od ciljeva testiranja su: [1]

### Verifikacija i validacija

Cilja testiranja nije samo pronaći pogreške u kodu ili dizajnu. Cilj je verificirati da software zaista radi ono što je namijenjen i kako je zamišljen. Jedan od rezultata testiranja je i izvještaj (*test report*).

### Prioretiziranje pokrivenosti

U idealnom svijetu sa neograničenim resursima svaki dio koda i funkcionalnosti bi bio pokriven testovima, ali nažalost to nije moguće. Zato je bitno pravilno odrediti što je prioritet te što će se pokriti testovima. U pravilu su to one funkcionalnosti i značajke koje nisu vidljive na prvi pogled čim se otvori program jer su takvi problemi lako uočljivi svakome. Isto tako, beskonačni testovi uzimaju mnogo dragocjenog vremena pa je i u tom pogledu bitno odrediti što se treba testirati.

### Sljedivost

Dokumentiranje testova kada se nešto i kako testiralo je bitno kako bi se u slučaju pojave problema moglo odrediti kada je i koja promjena uzrokovala neželjeno ponašanje proizvoda. To je posebno bitno u određenim kategorijama softwarea kao npr. u financijskom poslovanju gdje se može tražiti dodatna odgovornost samog proizvoda.

### 1.3 Uloga testera u timu

Glavna uloga testera, kao i samog procesa testiranja, je dodatna sigurnosna mreža koja je samo zadnja karika u lancu procesa testiranja i osiguranja kvalitete proizvoda. Dok su programeri (*developeri*, *engl. developers*) zaduženi za pisanje unit testova, testeri su zaduženi za pisanje i izvršavanje funkcionalnih testova. Testeri također pomažu product owneru sa izradom kriterija za uspješno prihvatanje rezultata testova [2]. Bitno je napomenuti kako su i developeri i testeri dio istog tima te kao tim imaju zajednički cilj - isporuka najkvalitetnijeg proizvoda moguće unutar zadanih parametara.

## Poglavlje 2

# Playwright

Playwright je open-source biblioteka za automatizaciju testiranja web preglednika i web skrapanja koju je razvio Microsoft. Omogućuje automatizaciju testiranja web aplikacija na Chromiumu, Firefoxu i WebKit-u s jednim API-jem.

Prednosti Playwrighta:

- Jednostavan za korištenje: Playwright ima intuitivan API koji je sličan JQuery-ju i Cypress-u.
- Brz i pouzdan: Playwright je optimiziran za brzinu i pouzdanost, što ga čini idealnim za testiranje web aplikacija u produkciji.
- Svestran: Playwright se može koristiti za testiranje različitih tipova web aplikacija, uključujući jednostavne web stranice, jednostruke web aplikacije (SPA) i višestruke web aplikacije (MPA).
- Podržava više jezika: Playwright se može koristiti s raznim jezicima programiranja, uključujući JavaScript, TypeScript, Python, Java i C#.

Playwright se može koristiti za:

- Automatizaciju UI testova: Playwright se može koristiti za pisanje automatiziranih UI testova koji provjeravaju funkcionalnost web aplikacija.
- Web skrapanje: Playwright se može koristiti za prikupljanje podataka sa web stranica.
- Generiranje screenshot-ova i videozapisa: Playwright se može koristiti za generiranje screenshot-ova i videozapisa web stranica.

## 2.1 Opis i pregled paketa

Sistemske zahtjevi za pokretanje Playwrighta su: <sup>1</sup>

- Node.js 18 ili noviji
- Windows 10 ili noviji, Windows Server 2016 ili noviji ili Windows Subsystem for Linux (WSL),

---

<sup>1</sup><https://playwright.dev/docs/intro#system-requirements>

- MacOS 12 Monterey ili noviji
- Debian 11, Debian 12, Ubuntu 20.04 ili Ubuntu 22.04, sa x86-64 ili arm64 arhitekturom.

Omogućava testiranje na Chromium, Firefox i WebKit enganima <sup>2</sup> koji se koriste u modernim web preglednicima.

Paket omogućava izvršavanje testova u UI načinu rada kao i u *headless* načinu rada prilikom kojeg se ne vide koraci kako se kreće po web stranici nego se na kraju testa dobije izvještaj o uspješnosti testiranja. To je vrlo korisno kada se koriste automatski načini objavljivanja koda koji onda može izvršiti testiranje prilikom svake promjene koda.

## 2.2 Instalacija

Najjednostavniji način za instalaciju Playwright paketa je putem npm alata koristeći naredbu

```
npm init playwright@latest
```

te će to instalirati paket i pokrenuti postupak inicijalizacije paketa. Osim **npm**, može se koristiti i **yarn** ili **pnpm**, ovisno o osobnim preferencijama. Tokom inicijalizacije može se birati nekoliko postavki:

- Odabrati TypeScript ili JavaScript (standardno je TypeScript)
- Odabrati ime direktorija koji će sadržavati testove (standardno je 'test' ili 'e2e' - end to end, ako 'test' već postoji)
- Dodati GitHub Action workflow za automatsko izvršavanje testova prilikom objave izvornog koda na GitHub servisu
- Instalirati potrebne preglednike koji će se koristiti za testiranje

Playwright će nakon toga kreirati potrebne direktorije i datoteke za konfiguraciju kao i primjer jednog testa za lakši početak

```
playwright.config.ts
package.json
package-lock.json
tests/
  example.spec.ts
tests-examples/
  demo-todo-app.spec.ts
```

Primjetimo kako je uvrijeđena norma da se datoteke koje sadrže testove imaju **.spec** ispred oznake tipa datoteke uz zadržavanje istog imena. Čak ih i razni editori koda označavaju s drugim ikonama kako bi bili vizualno lakše raspoznatljivi od datoteka koje sadrže izvorni komponenti kao što je vidljivo na slici 2.1.

Ukoliko se inicijalizacija vrši unutar već postojećeg projekta, što je najčešće i slučaj, konfiguracija zavisnih paketa će biti dodana u postojeću **package.json** datoteku.

**playwright.config.ts** datoteka sadrži konfiguracije testova kao npr

---

<sup>2</sup><https://www.npmjs.com/package/playwright#documentation--api-reference>



Slika 2.1: Izgled ikona sa izvornim kodom i testom za komponentu

- koji se preglednik koristi,
- koja je veličina prozora preglednika,
- koji se mobilni uređaj koristi u slučaju testiranja na mobilnim preglednicima,
- standardno očekivano vrijeme ispunjenja testa (timeout)

te mnogi drugi preddefinirane i prilagođene opcije konfiguracije.

Na slici 2.2 vidimo kako izgleda uspješna instalacija i inicijalizacija Playwright paketa.



```

> npm init playwright@latest
Getting started with writing end-to-end tests with Playwright:
Initializing project in '.'
✓ Do you want to use TypeScript or JavaScript? · TypeScript
✓ Where to put your end-to-end tests? · tests
✓ Add a GitHub Actions workflow? (y/N) · false
✓ Install Playwright browsers (can be done manually via 'npx playwright install')? (Y/n) · true
Initializing NPM project (npm init -y)...
Wrote to /Users/kristijancetina/Developer/jsTesting/report/tp/package.json:

{
  "name": "tp",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}

Installing Playwright Test (npm install --save-dev @playwright/test)...
added 4 packages, and audited 5 packages in 1s

found 0 vulnerabilities
Installing Types (npm install --save-dev @types/node)...
added 2 packages, and audited 7 packages in 617ms

found 0 vulnerabilities
Writing playwright.config.ts.
Writing tests/example.spec.ts.
Writing tests-examples/demo-todo-app.spec.ts.
Writing package.json.
Downloading browsers (npx playwright install)...
✓ Success! Created a Playwright Test project at /Users/kristijancetina/Developer/jsTesting/report/tp

Inside that directory, you can run several commands:

npx playwright test
  Runs the end-to-end tests.

npx playwright test --ui
  Starts the interactive UI mode.

npx playwright test --project=chromium
  Runs the tests only on Desktop Chrome.

npx playwright test example
  Runs the tests in a specific file.

npx playwright test --debug
  Runs the tests in debug mode.

npx playwright codegen
  Auto generate tests with Codegen.

We suggest that you begin by typing:

npx playwright test

And check out the following files:
- ./tests/example.spec.ts - Example end-to-end test
- ./tests-examples/demo-todo-app.spec.ts - Demo Todo App end-to-end tests
- ./playwright.config.ts - Playwright Test configuration

Visit https://playwright.dev/docs/intro for more information. 🌟

Happy hacking! 🚀

```

Slika 2.2: Ekran nakon uspješne instalacije i inicijalizacije Playwright paketa

## 2.3 Generiranje testova

Playwright ima mogućnost generiranja testova tako što snima klikanje miša korisnika te to prevodi u kod koji se može izvršavati. To je vrlo jednostavan i praktičan način da čak i totalni početnici mogu krenuti s izradom automatskih testova, a kasnije s

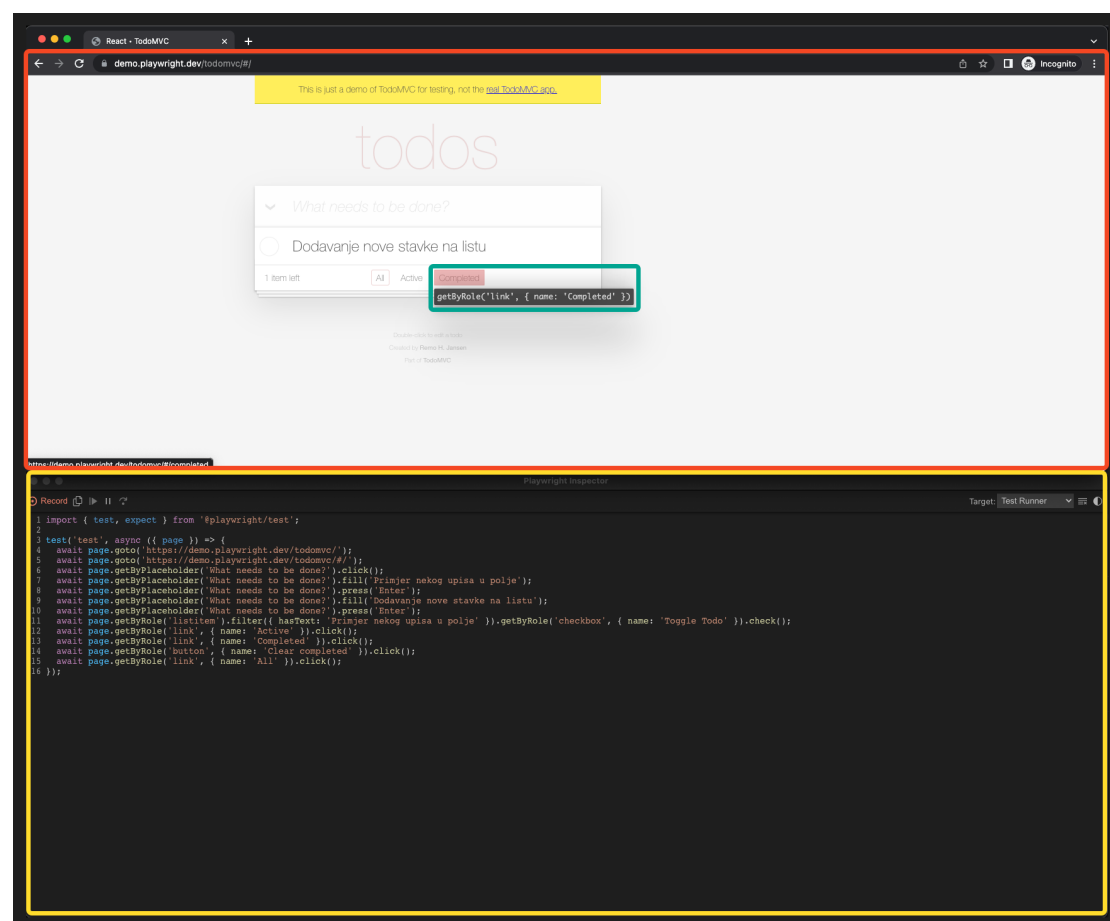
iskustvom se ti testovi mogu rafinirati i poboljšavati. Često je i tako generirai kod dovoljno dobar za upotrebu kod jednostavnijih slučajeva, a zasigurno je dobar za brzo sastavljanje testova da se izbjegne često dosatno tipkanje djelova koda koji se neće kasnije ponovno upotrebljavati. To uklanja potrebu za pisanje prilagođenih funkcija kao što radimo prilikom izrade projekta koji planiramo dugo vremena održavati i na taj način se može uštediti dosta vremena. Iako, treba biti oprezan s time jer često ušteda vremena na početku vodi do puno utrošenog vremena kasnije, ali to je tema koja je izvan okvira ovog rada.

## 2.4 Pokretanje alata za generiranje testova

Alat za generiranje testova se pokreće putem `codegen` naredbe koja prima argument URL web stranice za koju se želi generirati testovi. URL nije obavezan i može se pokrenuti alat bez njega, a zatim dodati URL izravno u prozoru preglednika.

Pokazati ćemo to na promjeru koji se nalazi u službenoj dokumentaciji <sup>3</sup> koristeći naredbu

```
npx playwright codegen demo.playwright.dev/todomvc
```



Slika 2.3: Izgled sučelja za generiranje testova

<sup>3</sup><https://playwright.dev/docs/codegen-intro#running-codegen>

Na slici 2.3 je prikazan izgled sučelja za generiranje testova koji se sastoji od nekoliko dijelova:

- prozor preglednika unutar kojeg se izvršava aplikacija - označen s crvenim okvirom (gore)
- prozor unutar kojeg se prikazuje generirani kod - označen s žutim okvirom (dolje)
- Lokator koji će Playwright koristiti se prikazuje kada se stavi miš preko elementa - označen sa zelenim okvirom (unutar prozora preglednika)

## **2.5 Kontinuirana integracija - CI**

## Poglavlje 3

# Testiranje nakon nadogradnje na novu verziju

Jedna od čestih aktivnosti svakog razvojnog tima je nadogradnja programa na novu verziju. U poduzeću je praksa da se pušta u produkcijski rad nova verzija programa jednom mjesečno za glavnog klijenta. Kako klijent ima postrojenja na 6 kontinenta uz dodatna postrojenja koja se nalaze u međunarodnim vodama svih svjetskih oceana proces nadogradnje je podijeljen po regijama. To su Americas (Sjeverna i Južna Amerika), EU (Europa i Bliski istok + Afrika) te AP (Azija i Pacifik). Svaka od regije ima više od 15 postrojenja koja koriste zasebne servere što znači da se nadogradnja vrši na više od 15 lokacija po regiji. Već iz ovoga je izvjesno kako je to puno potencijalnih problema i zastoja koja se mogu pojaviti te treba osigurati da je proces nadogradnje čim je više optimalan bez nepotrebnih zastoja i prekida u radu.

### 3.1 Postojeći način testiranja

Do uvođenja automatskih testova standardna procedura se sastojala od toga da AM regiju detaljno testira 3-4 testera (inženjera za kontrolu kvalitete - QA) koji bi svaki od njih provjerio 3 do 4 postrojenja (site). U prosjeku, za provjeriti jedno postrojenje je trebalo 20 do 30 minuta kako bi se osiguralo da su sve promjene aplicirane ispravno te da nisu izazvale neželjene nuspojave i prestanak rada postojećih funkcionalnosti. Naravno, zbog velikog broja funkcionalnosti te nedostatka vremena da se svaka od njih ručno provjeri bili smo vrlo izbirljivi što će se testirati s obzirom na kritičnost neke funkcionalnosti. Cijeli taj proces bi trajao oko 1:30 do 2 sata po članu test tima, a sve ukupno bi se trošak procijenio na 6-7 radnih čovjek-sati. Ako to pomnožimo s cijenom rada lako se dolazi do ukupnog troška testiranja nakon nadogradnje.

Nakon što se verificiralo da je novi paket programa ispravan, te ako klijenti nisu prijavili ozbiljne probleme koji ih sprječavaju u radu putem sustava za prijavu kvara (*showstopper*), moglo se pristupiti nadogradnji sljedeće regije Asia Pacific.

Tada bi se ponovila opet ista procedura, uz eventualne izmjene da bi bio 1 član testinog tima manje kako bi se smanjili prekovremeni sati jer nije bilo potrebe za provjerom svih funkcionalnosti, već je bilo prihvatljivo da se provjeri i reducirani set za koji bi trebalo maksimalno 20 minuta po postrojenju što bi ukupno iznosilo oko 5 čovjek-sati po regiji. Iako je to manji trošak, i dalje je značajni trošak. Čim više što se je termin za radove na AP regiji djelomično izvan redovnog radnog vremena poduzeća tako da treba uračanti i trošak prekovremenog rada. To je posebno izraženo za EU

regiju kod koje pak termin za radove je u potpunosti izvan redovnog radnog vremena pa je u tom slučaju sav rad je prekovremeni rad.

Uz sve navedeno dodatan problem ručnog testiranja je točnost i preciznost istog. Isto tako je normalno za očekivati kako će različiti tester i odraditi posao na različitoj razini kvalitete jer ipak nisu strojevi. Zato smo odlučili taj posao prepustiti strojevima. Barem u mjeri koliko je to moguće.

Iz gore navedenog je razvidno kako je bilo potrebno poboljšati proces testiranja nadogradnje primarno iz razloga povećanja kvalitete posla.

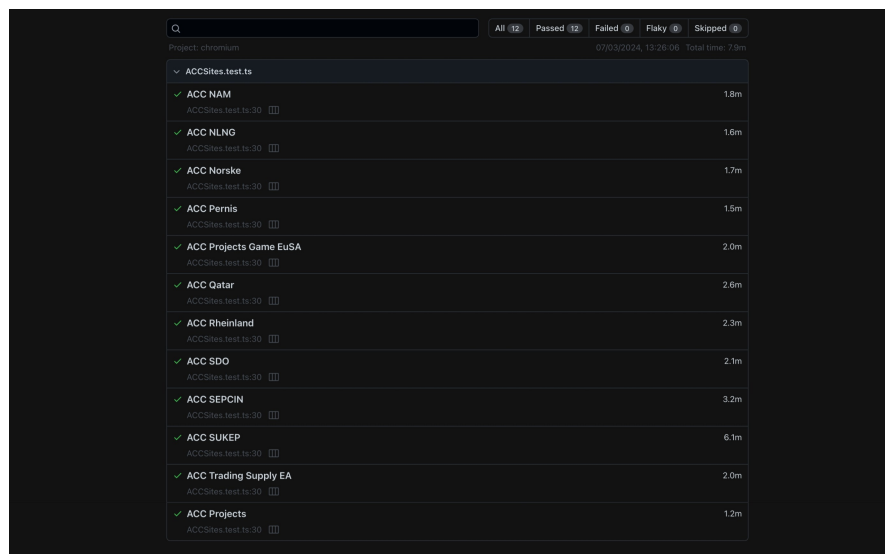
## 3.2 Automatsko testiranje procesa nadogradnje

Glavni cilj prelaska na automatski način testiranja je bio povećati preciznost testova i osigurati da se ne izostavi niti jedan korak koji se provjeravao pri ručnom testiranju.

Drugi cilj je bio smanjiti vrijeme potrebno za testiranje sa prethodnih 30 minuta na ispod 5 minuta.

Dodatan cilj je bio omogućiti daljni rast broja klijenata i instalacija koja se mogu nadograditi istovremeno, a bez da se mora uložiti značajno više vremena. Idealno smo htjeli postići  $\mathcal{O}(\log n)$  rast potrošenog vremena za svakog novog klijenta.

Nakon što je provedena nadogradnja s automatskim testiranjem potvrđeno je da su glavni ciljevi ostvareni te da će se nastaviti s implementacijom primjenjene tehnologije i za ostale potrebe. Konkretno, vrijeme testa po postrojenju je iznosilo od 2-4 minuta, ovisno o performansama servera i količini podataka koje klijent ima za razne izvještaje. Na slici 3.1 je prikazan izvještaj koji Playwright generira nakon završenog testiranja.



Test Name	Duration
ACC NAM	1.8m
ACC NLNG	1.6m
ACC Norske	1.7m
ACC Pernis	1.5m
ACC Projects Game EuSA	2.0m
ACC Qatar	2.6m
ACC Rheinland	2.3m
ACC SDO	2.1m
ACC SEPCIN	3.2m
ACC SUKEP	6.1m
ACC Trading Supply EA	2.0m
ACC Projects	1.2m

Slika 3.1: Izvještaj nakon završenog automatskog testiranja

Niti jedan test nije pokazivao znakove problema, ako ih zaista nije bilo (*false-negative* testovi). Svi uočeni problemi nakon isporuke nove verzije su bili riješeni u rekordnom roku, a tome je pridonjelo vrlo kratko vrijeme do otkrivanja problema.

Kao dodatan benefit je primjećeno da se sada tester i mogu više fokusirati na specifične rubne slučajeve koje je potrebno verificirati te na taj način dodatno

smanjiti broj prijavljenih nedostataka od strane korisnika te posljedično povisiti kvalitetu isporučenog proizvoda.

## Poglavlje 4

# Implementacija u produkciji na stvarnom proizvodu

Jedan od razloga za proučavanje ove teme je bio rješavanje stvarnog problema opisanog u 3.1. U nastavku ovog poglavlja biti će opisan izvorni kod koji se koristio za provedbu automatskog testiranja.

Prikazana skripta demonstrira organizirani pristup automatizaciji testiranja web aplikacija pomoću Playwrighta. Iskorištavanjem POM-ova i modularnih funkcija, osigurava da su testovi čitljivi i održavani. Svaka stranica iz JSON konfiguracije prolazi slijed provjera za provjeru funkcionalnosti, uz fleksibilnost da se prema potrebi jednostavno omogući ili onemogući određeni testni korak.

### 4.1 Uvođenje i inicijalizacija

Skripta započinje serijom uvoženja pomoćnih komponenti potrebnih za testiranje

```
import { test, expect, Page } from '@playwright/test';
import { MainGridToolbarPage } from '../pom/MainGridToolbar';
import { MainNavTabsPage } from '../pom/MainNavTabs';
import { FLOCFORM } from '../pom/FLOCFORM';
import { LoginPage } from '../pom/Login';
import { DevToolsPage } from '../pom/DevTools';
import { DashboardPage } from '../pom/Dashboard';
import { MainGridPage } from '../pom/MainGrid';
import { ReportsLightboxPage } from '../pom/ReportsLightbox';
import { FeedbackPage } from '../pom/Feedback';
import { CorrosionLoopPage } from '../pom/CorrosionLoop';
import { ModulePickerPage } from '../pom/ModulePicker';
import { BaseMainOptions } from '../pom/BaseMain';
import { MyAccount } from '../pom/MyAccount';
import { MainGridFiltersPage } from '../pom/MainGridFilters';
import sites from '../data/ListOfSites.json';
import { format } from 'date-fns/format';
import { enGB } from 'date-fns/locale';
```

Navedene komande uvoze djelove Playwright biblioteke potrebne za učitavanje stranice kao i provjeru dobivenih rezultata s očekivanim vrijednostima. Nadalje, cijeli

proizvod je podijeljen u manje komponente koje se dijele unutar programa te su isti definirani u objektnim modelima stranica (POM - page object module u nastavku teksta) Podaci o stranicama koje treba testirati su učitani putem JSON formata te također se koristi `date-fns` biblioteka za manipulaciju i formatiranje datuma.

## 4.2 Deklaracija varijabli

Sljedeće varijable su deklarirane za spremanje instanci POM klasa

```
let loginPage: LoginPage;
let modulePickerPage: ModulePickerPage;
let mainGridPage: MainGridPage;
let mainGridToolbarPage: MainGridToolbarPage;
let baseMain: BaseMainOptions;
let mainNavTabsPage: MainNavTabsPage;
let devTools: DevToolsPage;
let reportsLightboxPage: ReportsLightboxPage;
let myAccount: MyAccount;
```

## 4.3 Konfiguracija testova

Globalna konfiguracija testova je definirana u `/playwright.config.ts` datoteci, ali je moguće za svaki test definirati dodatne parametre ili nadvladati globalne kada je to potrebno. Za ovaj test je dodatno definirano da želimo da se izvršava paralelnom načinu izvođenja kako bi se poboljšale performanse s obzirom da nema potrebe da se čeka završetak jednog testa da bi se pokrenuo naredni. To je jednostavno napravljeno sljedećom linijom koda:

```
test.describe.configure({ mode: 'parallel' });
```

## 4.4 Glavni test

Glavna petlja koda koji se izvršava za svakog klijenta (site) je:

```
for (const site in sites) {
  const currentDate = format(new Date(), 'd MMM yyyy, HH:mm:ss', {
    locale: enGB,
  });
  const siteObj = sites[site];

  test(site, async ({ page }) => {
    const url: string = siteObj.url;
    // Instantiate POM classes with the current page instance
    loginPage = new LoginPage(page);
    modulePickerPage = new ModulePickerPage(page);
    mainGridPage = new MainGridPage(page);
    mainGridToolbarPage = new MainGridToolbarPage(page);
    baseMain = new BaseMainOptions(page);
    mainNavTabsPage = new MainNavTabsPage(page);
```



```

    devTools = new DevToolsPage(page);
    reportsLightboxPage = new ReportsLightboxPage(page);
    myAccount = new MyAccount(page);

    // Perform login and initial setup
    await test.step('Login Check', async () => {
        await checkConsoleLog(page, site);
        await page.goto(url + '#fcmfloc');
        await login(page);
        await mainGridToolbarPage.clearAllFilters();
        await mainGridToolbarPage.clearScoping();
    });

    await test.step('Feedback Check', async () => {
        const feedbackPage = new FeedbackPage(page);
        await feedbackPage.sendFeedback();
    });

    await test.step('HD Tool Check', async () => {
        await devTools.checkHDTool(url);
    });

    await test.step('Reset Survey Provider', async () => {
        // Must be called immediately after HD Tool Check,
        // until the page refresh is fixed.
        await devTools.resetSurveyProvider();
    });

    if (siteObj.PEI == true) {
        await test.step('Go To PEI', async () => {
            await modulePickerPage.goToPEI(url);
        });
    }

    await test.step('PEI Main Grids Check', async () => {
        await baseMain.goToMyAccount();
        await myAccount.checkPEIMainGrids();
    });

    await test.step('Checklist Findings Check', async () => {
        if (siteObj.CIVIL == true) {
            await modulePickerPage.selectCivil();
        }
        await checkChecklistFindings(page);
    });

    await test.step('PI Service Check', async () => {
        if (siteObj.PIService == true) {
            await checkPIService(page);
        }
    });

```

```

});

await test.step('Dynamic Forms Check', async () => {
  await checkDynamicForms(page);
});

    if (siteObj.FCM == true) {
      await test.step('Go To FCM', async () => {
        await modulePickerPage.goToFCM(url);
      });

      await test.step('FCM Main Grids Check', async () => {
        await baseMain.goToMyAccount();
        await myAccount.checkFCMMainGrids();
      });
    }

    if (siteObj.MobUrl) {
      await test.step('Mobile Site Check', async () => {
        await page.goto(siteObj.MobUrl + '#peifloc');
        await login(page);
        console.log('Mobile Site Check Successful');
      });
    } else {
      console.log(site + " doesn't have a mobile site url");
    }
  });
}

```

Unutar petlje se provjerava niz krucijalnih funkcionalnosti koji moraju raditi nakon svakog ažuriranja, a praksa je pokazala da ponekad, iz raznih razloga, to nije slučaj. Primarno su to logiranje u program, navigacija na specifične module, provjera logging alata i resetiranje analitičkih alata. Svaki korak testa je definiran kao asinkroni `await test.step` koji objedinjuje jedan korak.

## 4.5 Pomoćne funkcije

Više asinkronih pomoćnih funkcija je definirano koje sadrže specifične radnje koje se ponovno koriste u različitim koracima testiranja, poboljšavajući modularnost koda i mogućnost održavanja.

```

async function login(page: Page) {
  await loginPage.verifyVersion();
  await loginPage.loginCorpAdm();
  const okButton = page.locator("//button[text()='OK']");
  if (await okButton.isVisible()) {
    await okButton.click();
  }
  console.log('Login Check Successful');
}

```

## Poglavlje 5

# Zaključak

U ovom radu prikazan je postupak izrade

# Literatura

- [1] S. Quadri and S. U. Farooq, “Software testing—goals, principles, and limitations,” *International Journal of Computer Applications*, vol. 6, no. 9, p. 1, 2010.
- [2] A. Mundra, S. Misra, and C. A. Dhawale, “Practical scrum-scrum team: Way to produce successful and quality software,” in *2013 13th International Conference on Computational Science and Its Applications*, pp. 119–123, IEEE, 2013.
- [3] Microsoft Inc., “Playwright homepage.” <https://playwright.dev/>. (3.4.2022.).
- [4] B. J. Sauser, R. R. Reilly, and A. J. Shenhar, “Why projects fail? how contingency theory can provide new insights—a comparative analysis of nasa’s mars climate orbiter loss,” *International Journal of Project Management*, vol. 27, no. 7, pp. 665–679, 2009.
- [5] T. Linz, *Testing in scrum: A guide for software quality assurance in the agile world*. Rocky Nook, Inc., 2014.
- [6] R. Löffler, B. Güldali, and S. Geisen, “Towards model-based acceptance testing for scrum,” *Softwaretechnik-Trends*, GI, 2010.

# Popis slika

1.1	Proces testiranja u Scrum metodologiji rada . . . . .	3
2.1	Izgled ikona sa izvornim kodom i testom za komponentu . . . . .	8
2.2	Ekran nakon uspješne instalacije i inicijalizacije Playwright paketa . . . . .	9
2.3	Izgled sučelja za generiranje testova . . . . .	10
3.1	Izvještaj nakon završenog automatskog testiranja . . . . .	13