

UNIVERZA V LJUBLJANI  
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Jan Ivanetič

**Analiza orodij za kreiranje  
avtomatskih testov za odjemalce z  
govornim vmesnikom**

DIPLOMSKO DELO

UNIVERZITETNI ŠTUDIJSKI PROGRAM  
PRVE STOPNJE  
RAČUNALNIŠTVO IN INFORMATIKA

MENTOR: prof. dr. Marko Bajec

Ljubljana, 2023

To delo je ponujeno pod licenco *Creative Commons Priznanje avtorstva – Deljenje pod enakimi pogoji 2.5 Slovenija* (ali novejšo različico). To pomeni, da se tako besedilo, slike, grafi in druge sestavine dela kot tudi rezultati diplomskega dela lahko prosto distribuirajo, reproducirajo, uporabljajo, priobčujejo javnosti in predelujejo, pod pogojem, da se jasno in vidno navede avtorja in naslov tega dela in da se v primeru spremembe, preoblikovanja ali uporabe tega dela v svojem delu, lahko distribuira predelava le pod licenco, ki je enaka tej. Podrobnosti licence so dostopne na spletni strani [creativecommons.si](http://creativecommons.si) ali na Inštitutu za intelektualno lastnino, Streliška 1, 1000 Ljubljana.



Izvorna koda diplomskega dela, njeni rezultati in v ta namen razvita programska oprema je ponujena pod licenco GNU General Public License, različica 3 (ali novejša). To pomeni, da se lahko prosto distribuira in/ali predeluje pod njenimi pogoji. Podrobnosti licence so dostopne na spletni strani <http://www.gnu.org/licenses/>.

*Besedilo je oblikovano z urejevalnikom besedil L<sup>A</sup>T<sub>E</sub>X.*

**Kandidat:** Jan Ivanetič

**Naslov:** Analiza orodij za kreiranje avtomatskih testov za odjemalce z govornim vmesnikom

**Vrsta naloge:** Diplomaska naloga na univerzitetnem programu prve stopnje  
Računalništvo in informatika

**Mentor:** prof. dr. Marko Bajec

**Opis:**

V diplomskem delu preučite, kakšna orodja in programska okolja so na voljo za avtomatsko testiranje aplikacij. Osredotočite se na okolja, ki podpirajo aplikacije z govornim komunikacijskim vmesnikom. Izberite dve najbolj perspektivni okolji, ki to omogočata in jih podrobneje preučite ter opišite. Naredite primerjavo med njima in ju preskusite na praktičnih primerih.

**Title:** Analysis of tools for creating automated tests for clients with a voice interface

**Description:**

In your thesis, you will examine what tools and software environments are available for automated application testing. Focus on environments that support applications with a speech communication interface. Select two of the most promising environments that allow this and study and describe them in more detail. Compare them and test them with practical examples.



*Rad bi se zahvalil svojemu mentorju prof. dr. Marku Bajcu za odzivnost in napotke pri izdelavi diplomske naloge. Želim se zahvaliti svoji družini za neprekinjeno podporo in spodbudo skozi celoten proces pisanja diplomske naloge, saj so bili vedno tam, ko sem jih potreboval in so mi pomagali ohranjati pozitivno miselnost. Prav tako bi se rad zahvalil svojim kolegom, ki so mi s svojim znanjem in idejami pomagali pri izpolnjevanju ciljev te naloge ter mi dali spodbudo in motivacijo za izboljšanje svojih dosežkov.*



# Kazalo

Povzetek

Abstract

<b>1</b>	<b>Uvod</b>	<b>1</b>
<b>2</b>	<b>Pregled orodij za avtomatizirano testiranje</b>	<b>3</b>
2.1	Predstavitev orodij za avtomatizacijo . . . . .	3
2.2	Izbira Playwrighta in Seleniuma . . . . .	12
<b>3</b>	<b>Namestitev in uporaba Playwrighta</b>	<b>15</b>
3.1	Priprava okolja . . . . .	15
3.2	Pisanje testov . . . . .	16
3.3	Zvočno testiranje . . . . .	20
3.4	Izvajanje testov . . . . .	23
<b>4</b>	<b>Namestitev in uporaba Seleniuma</b>	<b>25</b>
4.1	Priprava okolja . . . . .	25
4.2	Pisanje testov . . . . .	26
4.3	Zvočno testiranje . . . . .	30
4.4	Izvajanje testov . . . . .	31
<b>5</b>	<b>Sklepne ugotovitve</b>	<b>33</b>
	<b>Celotna literatura</b>	<b>37</b>





# Seznam uporabljenih kratic

kratica	angleško	slovensko
<b>API</b>	Application Programming Interface	Aplikacijski programski vmesnik
<b>ATDD</b>	Acceptance Test Driven Development	Razvoj s testiranjem sprejemljivosti
<b>RPA</b>	Robotic Process Automation	Robotska avtomatizacija procesov
<b>HTTP/2</b>	Hypertext Transfer Protocol 2	Protokol za prenos hiperteksta
<b>WebRTC</b>	Web Real-Time Communication	Spletna komunikacija v realnem času
<b>WAV</b>	Waveform Audio File Format	Format zvočne datoteke valovne oblike
<b>AIFF</b>	Audio Interchange File Format	Format datotek za izmenjavo zvoka
<b>MP3</b>	MPEG-1 Audio Layer 3	Zvočna plast MPEG-1 3
<b>URL</b>	Uniform Resource Locator	Enotni iskalnik virov
<b>CSS</b>	Cascading Style Sheets	Kaskadne stilske podloge
<b>CLI</b>	Command Line Interface	Vmesnik z ukazno vrstico



# Povzetek

**Naslov:** Analiza orodij za kreiranje avtomatskih testov za odjemalce z govornim vmesnikom

**Avtor:** Jan Ivanetič

Diplomska naloga se osredotoča na zasnovo avtomatskega testiranja programske opreme, ki se je v zadnjih letih uveljavilo kot najbolj učinkovita metoda za odkrivanje napak in zagotavljanje optimalnega delovanja. Zato je postalo avtomatsko testiranje nepogrešljivo orodje v svetu razvoja programske opreme, saj je zamenjalo staromodno ročno testiranje. V nadaljevanju naslavlja izzive, s katerimi se srečujemo pri testiranju aplikacij z govornim vmesnikom ter predstavi ustrezna orodja in ogrodja za avtomatsko testiranje tovrstnih aplikacij. Med naštetimi sta nato izbrani dve orodji, in sicer Playwright ter Selenium, ki ju konkretno predstavimo in primerjamo.

Glavni cilj diplomske naloge je tako preveriti orodja za avtomatsko testiranje aplikacij z govornim vmesnikom. Poudarek je na enostavnosti, učinkovitosti ter hitrosti pisanja in izvajanja testov. Končni rezultat dela nam pove, katero orodje je najbolj primerno za testiranje aplikacij, ki zadostujejo našim potrebam.

**Ključne besede:** avtomatski testi, govorni vmesnik, Playwright, Selenium.



# Abstract

**Title:** Analysis of tools for creating automated tests for clients with a voice interface

**Author:** Jan Ivanetič

This thesis focuses on the concept of automated software testing, which has emerged in recent years as the most effective method for detecting faults and ensuring optimal performance. As a result, automated testing has become an indispensable tool in the world of software development, eclipsing old-fashioned manual testing. In the following, we address the challenges encountered when testing applications with a speech interface and present relevant tools and frameworks for automated testing of such applications. Two of these tools, Playwright and Selenium, are then selected for concrete presentation and comparison.

The main objective of the thesis is thus to examine tools for automatic testing of speech-enabled applications. The focus is on simplicity, efficiency and speed of writing and running tests. The final result of the work tells us which tool is the most suitable for testing applications that meet our needs.

**Keywords:** automated tests, speech interface, Playwright, Selenium.



# Poglavje 1

## Uvod

V zadnjih letih se je uporaba spleta izjemno razširila, s tem pa so se povečale tudi zahteve po visoko kakovostni in interaktivni spletni vsebini. Razvijalci spletnih aplikacij morajo zagotavljati učinkovito delovanje svojih izdelkov, ki morajo hkrati zagotavljati visoko uporabniško izkušnjo. Ena izmed ključnih metod za zagotavljanje visoke kakovosti spletnih aplikacij je avtomatsko testiranje. Gre za proces testiranja programske kode brez uporabe ročnih testov, kar pomeni, da se lahko hitreje in učinkoviteje preveri pravilno delovanje kode ter s tem zmanjšajo stroški in čas pri testiranju. Kljub temu pa avtomatsko testiranje avdio aplikacij predstavlja poseben izziv zaradi kompleksnosti in občutljivosti zvočnih podatkov.

V današnjem času je avtomatsko testiranje postalo ključno za zagotavljanje kakovosti spletnih aplikacij. Z njim se lahko učinkovito in hitro preveri pravilno delovanje programske kode, kar pa ni dovolj za avdio aplikacije. Te predstavljajo izziv zaradi kompleksnosti in občutljivosti zvočnih podatkov, kar omejuje uporabo tradicionalnih pristopov k avtomatskemu testiranju. Zato se bomo v tej diplomski nalogi posvetili raziskovanju posebnih pristopov in metod za avtomatsko testiranje aplikacij z govornim vmesnikom. Naš cilj bo razviti rešitev, ki bo omogočala kakovostno testiranje zvočnih podatkov, obenem pa zmanjšala stroške in čas, potreben za testiranje.

V nadaljevanju bomo v Poglavlju 2 izvedli pregled orodij za avtomatizirano testiranje avdio aplikacij, analizirali njihove prednosti in slabosti ter izbrali dve najbolj primerni rešitvi za naš konkreten primer. V Poglavlju 3 bodo predstavljene in opisane tehnologije ter knjižnice, uporabljene v nadaljnjih poglavjih. Sledita Poglavlji 4 in 5, v katerih predstavimo pripravo okolja, pisanje testov, zvočno testiranje in izvajanje testov za Playwright in Selenium. Za dosego cilja bomo izpostavili tudi zahteve in omejitve, ki jih moramo upoštevati pri razvoju rešitve. Na koncu bomo v Poglavlju 6 naredili primerjavo uporabljenih orodij in povzeli raziskavo.



## Poglavje 2

# Pregled orodij za avtomatizirano testiranje

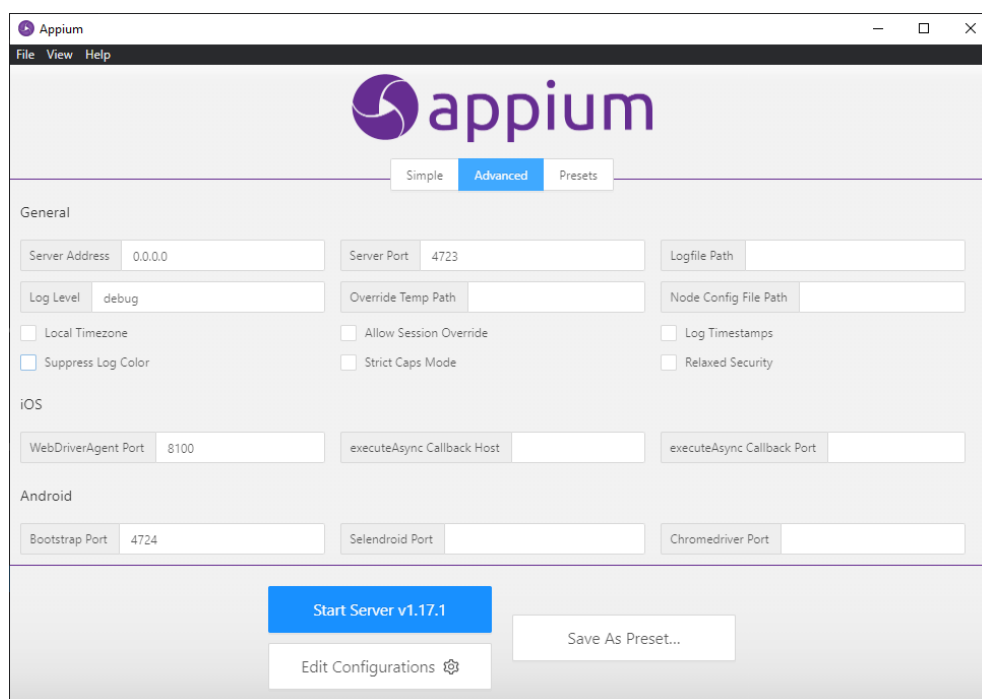
V poglavju bodo predstavljena orodja za avtomatsko testiranje aplikacij, na koncu pa bosta izmed naštetih izbrana dva, ki si ju bomo podrobneje pogledali v nadaljnjih poglavjih. Najprej bodo opisana primerna orodja za testiranje tako spletnih kot namiznih aplikacij, pri čemer bo poseben poudarek na preverjanju podpore za odjemalce z govornim vmesnikom. Nato bomo izbrali dve in utemeljili našo izbiro za nadaljnje delo z njima. Cilj tega poglavja je tako izmed naštetih izbrati dve, ki podpirata kreiranje avtomatskih testov za odjemalce z govornim vmesnikom.

## 2.1 Predstavitev orodij za avtomatizacijo

### 2.1.1 Appium

Odprtokodno orodje za avtomatizacijo Appium se uporablja za testiranje mobilnih aplikacij v več operacijskih sistemih, vključno z iOS in Androidom. Omogoča pisanje testov v različnih programskih jezikih, vključno z jeziki Java, Python in Ruby, ter podpira testiranje nativnih, hibridnih in mobilnih spletnih aplikacij. Grafični uporabniški vmesnik si lahko pogledamo na sliki 2.1, uspešno zagnan testni strežnik pa na sliki 2.2. Appium

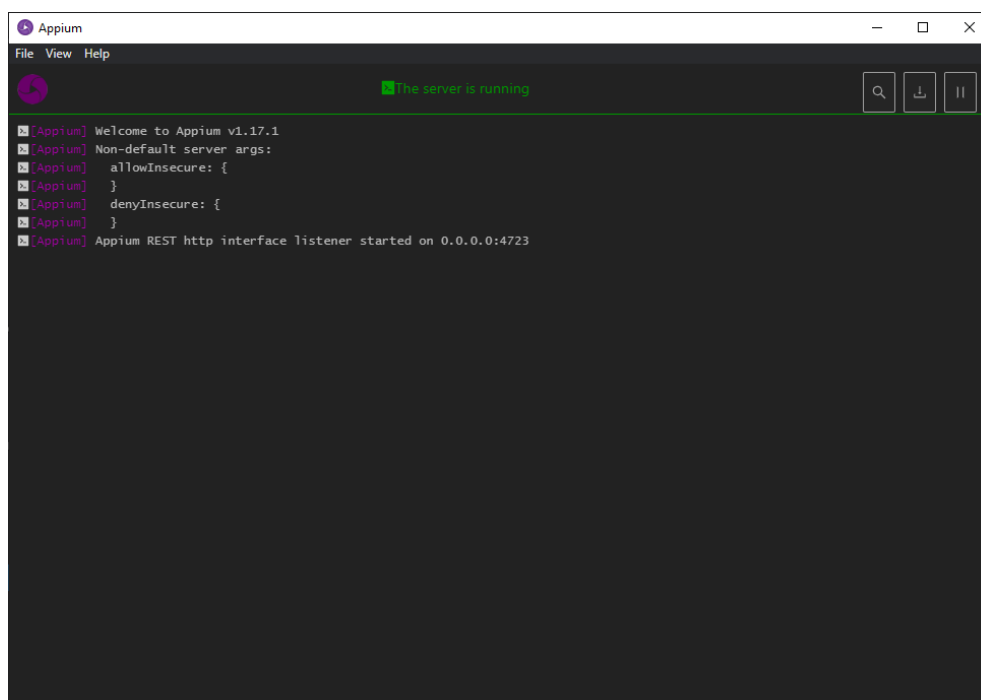
lahko poleg običajne uporabe pri funkcionalnostih uporabniškega vmesnika in uporabniških tokovih uporabljamo tudi za testiranje zvoka. Preizkušanje zvoka vključuje zagotavljanje, da funkcionalnosti aplikacije, povezane z zvokom, kot so prepoznavanje glasu, pretvorba besedila v govor in predvajanje zvoka, delujejo, kot je bilo predvideno [24].



Slika 2.1: Appium grafični uporabniški vmesnik

Appium z interakcijo z zvočnimi API-ji v mobilni napravi omogoča ocenjevanje funkcionalnosti zvoka. To pomeni, da je mogoče izdelati teste, s katerimi se ugotovi, ali program uspešno sprejema, obdeluje in oddaja zvok prek zvočnikov naprave. Omogoča uporabo knjižnic tretjih oseb, kot sta AudioUnit za iOS in OpenSL ES za Android, kar olajša testiranje zvoka. Te knjižnice omogočajo, da Appium komunicira z avdio podsistemom naprave ter simulira različne zvočne vhode in izhode, kar omogoča testiranje različnih avdio scenarijev [26].

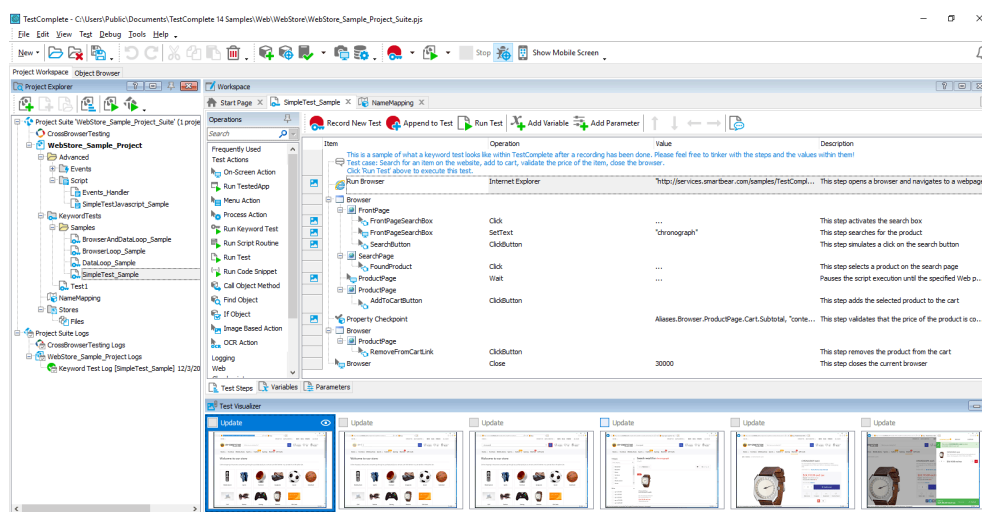
Na splošno je Appium prilagodljiva rešitev za testiranje mobilnih aplikacij, saj podpira testiranje zvoka in omogoča temeljito testiranje tako funkcionalnosti uporabniškega vmesnika kot tudi vidikov, povezanih z zvokom.



Slika 2.2: Uspešno zagnan Appium testni strežnik

### 2.1.2 TestComplete

TestComplete je zmogljivo orodje za avtomatizirano testiranje, ki ga je razvilo podjetje SmartBear Software in testerjem programske opreme pomaga pri ustvarjanju in izvajanju avtomatiziranih testov za namizne, spletne in mobilne aplikacije. Ponuja celovito rešitev za testiranje, ki se povezuje z več razvojnimi okolji, kot sta Visual Studio in Eclipse, ter podpira številne programske jezike, kot so Python, JavaScript in VBScript. Grafični uporabniški vmesnik si lahko ogledamo na sliki 2.3.



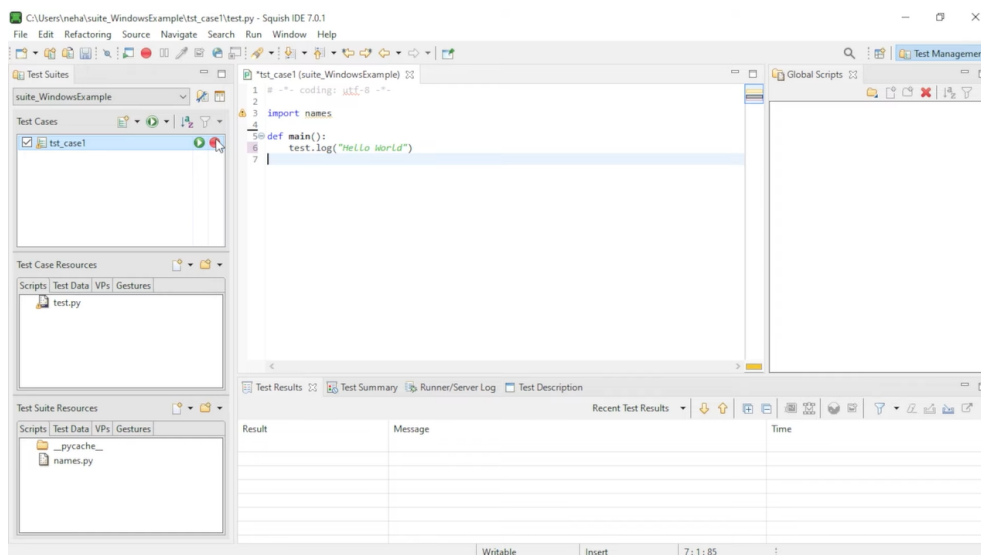
Slika 2.3: TestComplete grafični uporabniški vmesnik

Podpora za testiranje zvoka je ena od izstopajočih funkcij programa TestComplete. To je še posebej koristno za programe, kot so medijski predvajalniki, programska oprema za snemanje in urejanje zvoka ter programska oprema za prepoznavanje govora, ki temeljijo predvsem na zvoku. Z uporabo funkcij testiranja zvoka programa TestComplete lahko testerji avtomatizirajo teste, ki temeljijo na zvoku, kot so preverjanje kakovosti zvoka, predvajanje, snemanje in sinhronizacija. Testerji lahko v svoje testne skripte uvozijo zvočne datoteke ali snemajo zvočne testne skripte z vgrajenim snemalnikom zvoka TestComplete in ga nato uporabijo za testiranje zvoka. Zvočni izpis lahko nato primerjamo s predvidenimi rezultati z uporabo funkcij za predvajanje in analizo zvoka. Poleg tega TestComplete ponuja orodja za snemanje zvočnega vnosa za analizo in za simulacijo zvočnega vnosa, kot je na primer prepoznavanje glasu. [19]

Tako je TestComplete močan instrument za testiranje, ki ga lahko uporabimo za avtomatizacijo testiranja zvoka. Je prilagodljivo orodje za testiranje programske opreme, saj podpira veliko različnih programskih jezikov in se povezuje z različnimi razvojnimi okolji. Je uporabno orodje za vsako ekipo za testiranje zaradi svojih funkcij za testiranje zvoka, ki lahko testerjem pomagajo pri izboljšanju kakovosti in zanesljivosti aplikacij.

### 2.1.3 Squish

Squish, ki ga je razvilo podjetje Froglogic, je temeljita rešitev za testiranje, ki ponuja ogrodje za avtomatizirano testiranje več vrst programskih aplikacij, vključno z namiznimi, spletnimi in mobilnimi. Squish omogoča povezavo s številnimi razvojnimi okolji, vključno z Eclipse, Microsoft Studio in Qt Creator, ter testerjem omogoča, da sestavijo in izvajajo avtomatizirane teste z uporabo skriptnih jezikov, kot so Python, JavaScript in Ruby. Enostaven test znotraj Squish uporabniškega grafičnega vmesnika in v jeziku Python lahko vidimo na sliki 2.4.



Slika 2.4: Squish grafični uporabniški vmesnik

Poleg tega Squish podpira testiranje zvoka, kar uporabnikom omogoča samodejno testiranje programov, ki temeljijo na zvoku, vključno z medijskimi predvajalniki, programi za snemanje in urejanje zvoka ter programi za prepoznavanje govora. S pomočjo Squishovih funkcij za testiranje zvoka lahko preizkuševalci v svojih samodejnih testih snemajo, predvajajo in analizirajo vhodni in izhodni zvok ter se tako prepričajo, da program deluje, kot je bilo predvideno. Testerji lahko v svoje testne skripte naložijo zvočne datoteke ali snemajo zvočne vhodne podatke z vgrajenim snemalnikom zvoka, če ga

uporabljajo za testiranje zvoka v programu Squish. Zvočni izhod lahko nato primerjajo s predvidenimi rezultati z uporabo funkcij Squishovega predvajanja in analize zvoka. Squish ponuja tudi orodja za snemanje zvočnega vnosa za analizo in za posnemanje zvočnega vnosa, kot je prepoznavanje govora [13].

Zmožnost Squisha, da se poveže z različnimi orodji in ogrodji za testiranje, je ena od glavnih prednosti njegove uporabe za testiranje zvoka. Na primer, z integracijo Squisha z zelo priljubljenim ogrodjem za testiranje Robot Framework lahko preizkuševalci hitro razvijejo in izvajajo teste, ki temeljijo na zvoku, z uporabo sintakse ogrodja Robot Framework.

Squish je temeljito orodje za testiranje, ki testerjem omogoča avtomatizacijo testiranja aplikacij, ki temeljijo na zvoku. Zaradi zmožnosti integracije z drugimi ogrodji za testiranje je učinkovito orodje za testiranje programske opreme, njegova podpora za testiranje zvoka pa testerjem omogoča izboljšanje zmogljivosti in zanesljivosti aplikacij.

### **2.1.4 Robot Framework**

Robot Framework je odprtokodno ogrodje za avtomatizacijo testiranja, ki se pogosto uporablja za sprejemno testiranje, razvoj, ki temelji na sprejemnem testiranju (ATDD), in avtomatizacijo robotskih procesov (RPA). Napisan je v jeziku Python in omogoča ustvarjanje testnih primerov z uporabo ključnih besed (angl. keywords). Da bi se prilagodilo različnim zahtevam testiranja, ogrodje ponuja nabor knjižnic in orodij, ki jih je mogoče dopolniti s knjižnicami po meri. Enostavna integracija ogrodja Robot Framework z drugimi orodji in tehnologijami je ena njegovih glavnih prednosti. Uporablja se lahko na primer za testiranje API-jev, namiznih aplikacij, mobilnih aplikacij in spletnih aplikacij. Uporablja se lahko tudi za avtomatizacijo dela v različnih panogah, vključno s financami, zdravstvom in maloprodajo. Poleg tega ogrodje podpira različne vrste testiranja, kot so testiranje enote, funkcionalno testiranje in sprejemno testiranje [14].

Robot Framework se lahko uporablja za avtomatizacijo številnih operacij, povezanih z zvokom, vključno s predvajanjem zvoka, snemanjem zvoka in analizo zvoka, kar zadeva testiranje zvoka. Ogradje lahko razširimo z edinstvenimi knjižnicami, ki ponujajo ključne besede, povezane z zvokom, da bi dosegli cilj. Knjižnici SoundLibrary in SoundSpectrumLibrary ogradja Robot Framework se lahko na primer uporabljata za predvajanje oziroma analizo zvoka. Ti knjižnici se lahko uporabljata za preverjanje kalibra zvoka, iskanje nepravilnosti v zvoku in zagotavljanje skladnosti zvoka z industrijskimi standardi. Poleg tega je mogoče zaradi prilagodljivosti ogradja Robot Framework testiranje zvoka kombinirati z drugimi vrstami testiranja, kot sta funkcionalno in vizualno testiranje. To omogoča, da je testiranje zvoka del večjega in temeljitejšega načrta testiranja, ki preverja vse vidike aplikacije ali izdelka [20].

Robot Framework je prilagodljivo ogradje za avtomatizacijo testiranja, ki podpira različne vrste testiranja in ga je mogoče prilagoditi posebnim zahtevam testiranja. Je odlična možnost za testiranje zvoka, saj ponuja ključne besede, pomembne za zvok, in ga je mogoče kombinirati z drugimi metodami testiranja, da se oblikuje celovit pristop k testiranju.

### 2.1.5 Cypress

Cypress je priljubljeno ogradje za celostno testiranje, ki se uporablja za testiranje spletnih aplikacij. Razvijalci lahko ustvarijo teste, ki posnemajo interakcije uporabnikov s programom, in potrdijo, da program deluje, kot je bilo predvideno, saj ponuja celovito rešitev za testiranje. Cypressu je priložen zanesljiv izvajalec testov, ki lahko izvaja teste v realnem času, ko jih razvijalci napišejo, ponuja pa tudi enostaven in jasen API, ki razvijalcem omogoča preprosto izdelavo testov [3].

Cypress je mogoče prilagoditi tako, da podpira testiranje, ki je povezano z zvokom, čeprav ni bil ustvarjen z mislijo na testiranje zvoka. Ena od metod za to je uporaba vtičnikov in knjižnic s funkcijami za testiranje zvoka. Vtičnik AudioLabs/cypress-audio-controls na primer ponuja testna orodja,

povezana z zvokom, ki jih je mogoče povezati s testi Cypress, kot sta predvajanje zvočnih datotek in zaznavanje zvočnih dogodkov. Primerljive možnosti za anotacijo in vizualizacijo zvoka ponuja knjižnica sonic-annotate, ki jo je mogoče uporabiti za testiranje zvoka. Prilagodljiva arhitektura Cypressa prav tako omogoča nemoteno integracijo zvočnega testiranja z drugimi vrstami testiranja. Cypress se lahko na primer uporablja za ocenjevanje ne le uporabniškega vmesnika, funkcionalnosti in zmogljivosti spletne aplikacije, temveč tudi njenih funkcij, ki temeljijo na zvoku. S tem lahko pomagamo zagotoviti, da so z zvokom povezane funkcionalnosti aplikacije pravilno integrirane in delujejo, kot je bilo predvideno [6].

Če povzamemo, je Cypress zmogljivo ogrodje za testiranje, ki se lahko uporablja za testiranje spletnih aplikacij in ga je mogoče razširiti tako, da omogoča testiranje zvoka. Njegova prilagodljiva arhitektura omogoča povezovanje avdio testiranja z drugimi vrstami testiranja, vključuje pa tudi številne vtičnike in knjižnice, ki jih je mogoče uporabiti za avdio testiranje.

### 2.1.6 Playwright

Playwright je odprtokodno orodje za avtomatizacijo, ki je zasnovano za testiranje spletnih aplikacij v več brskalnikih in platformah. Celostno testiranje, funkcionalno testiranje in testiranje zmogljivosti so le nekatere od številnih vrst testiranja, za katere se lahko uporablja. Ponuja močan in razširljiv API za avtomatizacijo interakcij s spletnimi aplikacijami. Posebnost programa Playwright je njegova sposobnost modeliranja vedenja uporabnikov v številnih kontekstih brskalnika, vključno z več zavihki ali okni in različnimi napravami [12].

Playwrightove možnosti testiranja zvoka razvijalcem omogočajo avtomatizirano testiranje funkcij predvajanja in snemanja zvoka v spletnih aplikacijah. Razvijalci lahko s pomočjo vmesnika Web Audio API preizkušajo različne funkcije, povezane z zvokom, kot so predvajanje zvočnih datotek, prilagajanje zvočnih parametrov in zajemanje interakcije uporabnikov. S Playwrightovim samodejnim testiranjem zvoka je mogoče najti težave, vključno z zamikom



zvoka, popačenjem in napačno hitrostjo predvajanja. Poleg tega lahko zagotovi, da so zvočne funkcije dosledne v različnih napravah in brskalnikih, kar izboljša splošno uporabniško izkušnjo. Razvijalci lahko z uporabo programa Playwright za testiranje zvoka prihranijo čas in trud ter hkrati izboljšajo splošno kakovost svojih spletnih aplikacij [27].

Playwright je temeljito orodje za celostno testiranje, ki razvijalcem omogoča ocenjevanje uporabnosti, kakovosti zvoka in funkcionalnosti spletnih aplikacij. Zaradi svoje prilagodljivosti in preproste uporabe je dragocen dodatek k orodju vsake razvojne ekipe.

### 2.1.7 Selenium

Selenium je priljubljeno odprtokodno orodje za testiranje, ki se uporablja za avtomatizacijo spletnih aplikacij v različnih brskalnikih in platformah. Ponuja širok nabor zmožnosti, vključno z možnostjo simulacije interakcij z uporabniki, podporo številnim programskim jezikom in zanesljivim vmesnikom API [7].

Selenium je znan predvsem po svojih zmogljivostih za testiranje uporabniškega vmesnika, vendar ga lahko s podporo za vmesnik API Web Audio uporabljamo tudi za testiranje zvoka. Razvijalci lahko z uporabo programa Selenium avtomatizirajo testiranje zvočnih funkcij, kot so predvajanje, snemanje in upravljanje zvočnih lastnosti. Z avtomatiziranim testiranjem zvoka, ki temelji na programu Selenium, je mogoče najti težave, kot so popačenje zvoka, zakasnitev in težave s sinhronizacijo. Poleg tega lahko avtomatsko testiranje zvoka pomaga povečati dostopnost spletnih aplikacij. Za slabovidne uporabnike, ki za brskanje po spletnih straneh uporabljajo podporno tehnologijo, kot so bralniki zaslona, je zvok ključna sestavina. Razvijalci lahko z uporabo programa Selenium za testiranje zvočnih funkcij zagotovijo, da spletne aplikacije upoštevajo smernice za dostopnost in nudijo prijazno uporabniško izkušnjo za vse uporabnike [4].

Zaključimo lahko, da funkcije testiranja zvoka v programu Selenium razvijalcem nudijo številne prednosti, saj poenostavijo postopek testiranja in

izboljšajo kakovost zvočnih funkcij v spletnih aplikacijah. Z avtomatizacijo testiranja zvoka s programom Selenium lahko razvijalci odkrijejo težave v zgodnji fazi razvojnega cikla in zmanjšajo verjetnost, da bodo napake in pomanjkljivosti prišle v končni izdelek.

## 2.2 Izbira Playwrighta in Seleniuma

Pri odločitvi, katero orodje za avtomatizacijo testiranja spletnih aplikacij uporabiti, je pomembno upoštevati več faktorjev. Eden izmed njih je aktivna skupnost in redne posodobitve, saj omogočajo, da se orodje razvija v smeri potreb razvijalcev in spreminjajočih se tehnoloških trendov. Poleg tega je pomembno, da orodje podpira sodobne aplikacije in okolja ter omogoča avtomatizacijo testiranja govornega vmesnika.

Selenium je dobro znano orodje za avtomatizacijo testiranja spletnih aplikacij, ki ga uporablja veliko število razvijalcev. Zagotavlja preprosto in učinkovito avtomatizacijo testiranja spletnih aplikacij, vendar pa ima nekatere omejitve. Eno izmed njih so počasni časi odziva, kar lahko povzroča težave pri izvajanju testov na spletnih straneh, ki imajo veliko dinamičnih elementov. Poleg tega so lahko dinamični elementi težavni za zajemanje, kar lahko povzroči težave pri ustvarjanju robustnih testov. Selenium prav tako podpira naše zahteve za testiranje aplikacij z govornim vmesnikom, saj uporablja Chromiumove WebRTC funkcionalnosti.

Playwright je novo orodje za avtomatizacijo testiranja spletnih aplikacij, ki se hitro razvija in pridobiva na popularnosti zaradi svoje enostavnosti uporabe, hitrega izvajanja in učinkovitega zajemanja dinamičnih elementov na spletni strani. Prav tako ponuja podporo za več brskalnikov kot Selenium in ima aktivno in rastočo skupnost, kar zagotavlja odlično podporo in razvoj novih funkcionalnosti. Poleg tega tudi Playwright podpira avtomatizacijo testiranja govornega vmesnika s pomočjo Chromiumovih WebRTC zastavic.

Glede na zgoraj omenjene faktorje smo se odločili, da bomo v nadaljevanju dela uporabljali Selenium in Playwright v kombinaciji s Pythonom. Obe orodji imata aktivno skupnost, redne posodobitve ter sta tesno povezani z modernimi aplikacijami in okolji. Poleg tega sta primerna tudi za avtomatizacijo testiranja govornega vmesnika, kar nam omogoča, da lahko ustvarimo celovite teste, ki zajamejo vse vidike testiranja spletnih aplikacij. Z izbiro obeh orodij si zagotavljamo najboljše od obeh svetov in lahko izkoristimo njune prednosti pri ustvarjanju robustnih in zanesljivih testov.



## Poglavje 3

# Namestitev in uporaba Playwrighta

V poglavju bodo predstavljene možnosti priprave Playwright okolja v različnih programskih jezikih, pisanje testov ter pospešitev postopka s pomočjo vgrajene možnosti, testiranje aplikacije z govornim vmesnikom in izvajanje napisanih testov. Uporabljeni koda in primeri testov bodo na voljo na Github repozitoriju [5].

### 3.1 Priprava okolja

Playwright omogoča pisanje testov v več programskih jezikih, med njimi sta najbolj popularna Javascript in Python, podprti pa so tudi Typescript, C# in Java. Za namene diplomske naloge bo v nadaljnjih odsekih uporabljen programski jezik Python, predvsem zaradi enostavne namestitve, pisanja testov in izvajanja le-teh. Uporabljal se bo operacijski sistem Windows 10.

Za začetek priprave okolja je potrebno na sistem namestiti Python 3, vtičnik Playwright Pytest ter brskalnike Playwright, v katerih se bodo izvajali testi. Poleg tega bomo za izvedbo testov uporabili tudi tri interne Python knjižnice – wave, time in os.

Knjižnica wave [25] je zasnovana za branje in pisanje zvočnih datotek v

številnih formatih, kot so WAV, AIFF in MP3, in ponuja preprost vmesnik za delo z zvočnimi podatki. V našem primeru jo bomo uporabili za izračun dolžine zvočnega posnetka med predvajanjem.

Knjižnica `time` [22] ponuja številne funkcije za upravljanje s časom in sorodnimi vrednostmi. Za naše namene bomo uporabili funkcijo `sleep()`, ki omogoča ustavitev izvajanja programa za določeno število sekund.

Knjižnica `os` [8] pa je vgrajena knjižnica v Pythonu, ki nam omogoča interakcijo z operacijskim sistemom. V našem primeru jo bomo uporabili za sestavljanje poti do zvočnih posnetkov.

Poleg Python knjižnic bomo za izvedbo testov uporabili tudi Chromium WebRTC zastavice. WebRTC, kar pomeni spletno komuniciranje v realnem času, je tehnologija, ki omogoča neposredno izmenjavo avdio, video in podatkov med spletnimi brskalniki ter drugimi aplikacijami za medsebojno povezovanje. WebRTC je vgrajen v večino sodobnih spletnih brskalnikov in zato zelo preprost za uporabo ter široko dostopen. Uporablja nabor tehnologij, ki vključujejo zvočne in video kodeke, šifriranje in omrežne protokole, s čimer zagotavlja varno in zanesljivo komunikacijo. S tem omogoča enostavno ter učinkovito izvajanje testov za preverjanje delovanja spletnih aplikacij, ki uporabljajo govorni vmesnik.

Za bolj točen postopek namestitve knjižnic in priprave okolja se sklicujemo na Playwright Python dokumentacijo [10] ali opis repozitorija na Githubu [5].

## 3.2 Pisanje testov

Playwright trditve (angl. `assertions`) so ustvarjene posebej za dinamični splet. Preverjanja se samodejno ponavljajo, dokler niso izpolnjeni potrebni pogoji. Orodje ima prav tako vgrajeno funkcijo samodejnega čakanja, kar pomeni, da počaka, da so elementi strani uporabni pred izvedbo akcije nad njimi. Za testiranje trditev ponuja posebno funkcijo `expect`, ki počaka, da se nad lokatorjem izpolni pričakovan pogoj, definiran iz strani izvajalca testov.

### 3.2.1 Locator

Lokatorji ali iskalniki so ključni pari samodejnemu čakanju in možnosti ponovnega poskusa pri Playwrightu. Predstavljajo način iskanje elementov na spletni strani v vsakem trenutku in se uporabljajo za izvajanje akcij nad elementi in v kombinaciji s funkcijo `expect`. Najbolj pogosto uporabljeni vgrajeni lokatorji so [11]:

- `page.get_by_role()` za iskanje po eksplicitnih in implicitnih atributih;
- `page.get_by_text()` za iskanje po vsebini besedila;
- `page.get_by_label()` za iskanje elementa obrazca;
- `page.get_by_placeholder()` za iskanje vnosa po nadomestnem besedilu;
- `page.get_by_alt_text()` za iskanje elementa, običajno slike, po alternativnem besedilu;
- `page.get_by_title()` za iskanje elementa po naslovu

Ko enkrat s pomočjo lokatorja najdemo iskan element, lahko uporabimo množico funkcij, ki izvedejo nad elementom akcijo. Najbolj pogoste funkcije so sledeče:

- `fill()` za vstavljanje besedila v polje;
- `click()` za pritisk na element;
- `check()` za obkljukanje izbirnega polja;
- `nth()` za pridobitev n-tega rezultata poizvedbe;
- `filter()` za dodatno filtriranje rezultatov poizvedbe;
- `count()` za število rezultatov poizvedbe;
- `hover()` za premik nad elementov.

### 3.2.2 Expect

Playwright ponuja funkcijo `expect`, ki počaka, da se izpolni pričakovani pogoj. V kombinaciji z lokatorjem lahko funkcijo `expect` uporabimo na več načinov, najpogostejši so sledeči: [9]

- `expect(page).to_have_title()` da ima stran naslov;
- `expect(page).to_have_url()` da ima stran URL naslov;
- `expect(locator).to_have_attribute()` da bo atribut strogo enak vrednosti;
- `expect(locator).to_have_text()` da element vsebuje besedilo;
- `expect(locator).to_have_css()` da ima element CSS lastnost.

### 3.2.3 Enostaven primer Playwright testa

Primer Playwright testa, ki uporablja nekaj izmed prej naštetih funkcionalnost in funkcij, lahko vidimo v spodnji psevdokodi.

```
from playwright.sync_api import Playwright, expect

def test_prijava(playwright: Playwright) -> None:
    page.goto("https://accounts.google.com/")
    page.get_by_placeholder("email").fill("uporabnikov email")
    page.get_by_placeholder("password").fill("uporabnikovo geslo")
    page.get_by_role("button", name="PRIJAVA").click()
    expect(page).to_have_title("Gmail")
    expect(page).to_have_url(".*inbox")
```



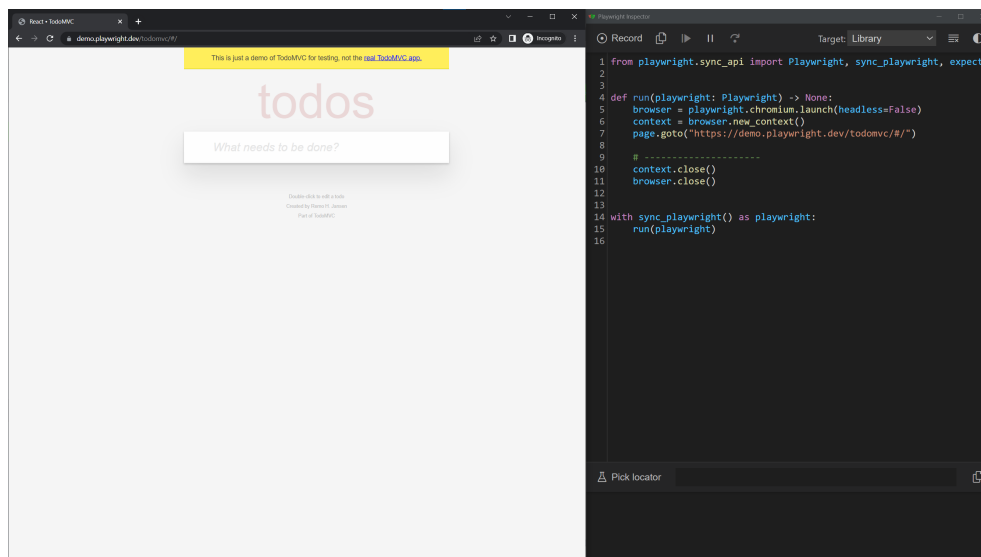
Prikazan primer nas pelje na Googlovo stran za prijavo, kjer vnesemo svoje uporabniško ime in geslo ter se prijavimo. Ali je bil postopek izveden pravilno, preverimo tako, da pogledamo, če je naslov strani Gmail in ali se v URL naslovu nahaja niz inbox.

### 3.2.4 Generiranje testov

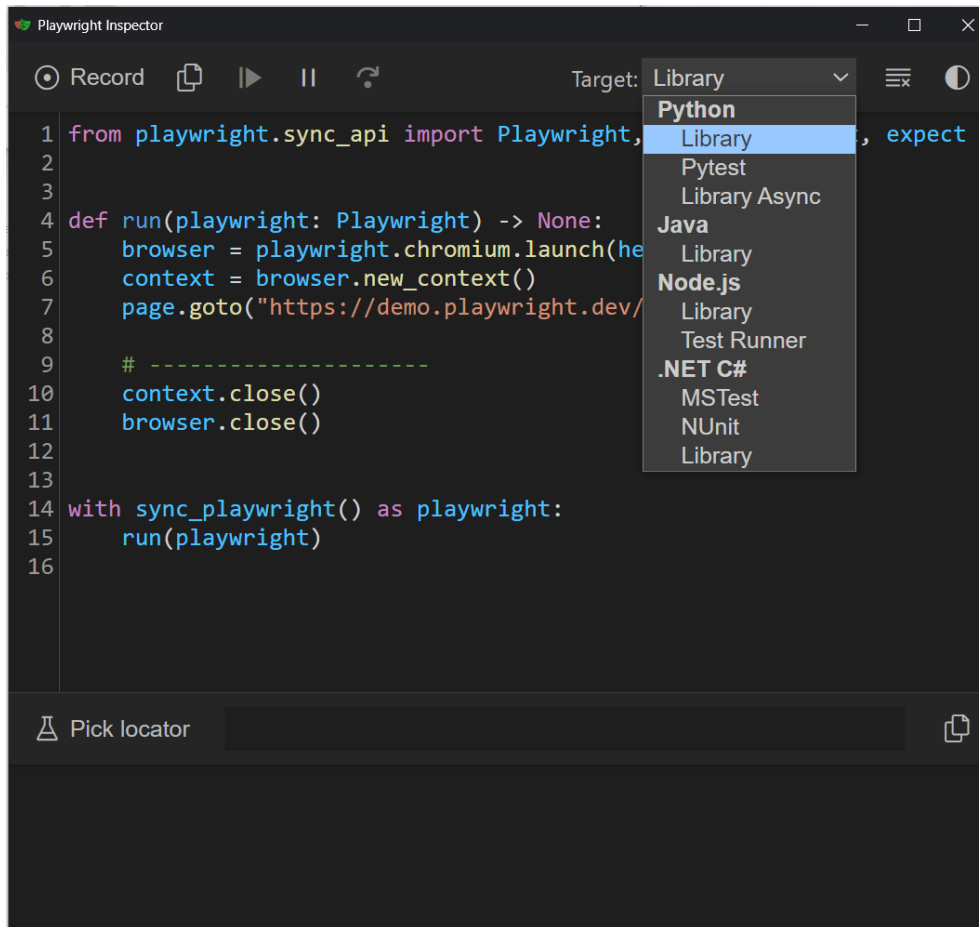
Playwright je opremljen z možnostjo takojšnjega ustvarjanja testov in je odličen način za hiter začetek testiranja. Za uporabo možnosti začnemo z izvedbo ukaza:

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

ki nam odpre dve okni na sliki 3.1. Eno je okno brskalnika, v katerem interaktiramo s spletnim mestom, ki ga želimo testirati in je enak kot navadna seja Chromovega brskalnika. Drugo pa je Playwright Inspector, v katerem se prikažejo izvedene akcije v brskalniku. Te lahko kopiramo, brišemo ali pa spreminjamo programskih jezik, ki ga želimo uporabljati, kot je razvidno iz slike 3.2.



Slika 3.1: Generiranje testov Playwright



Slika 3.2: Programski jeziki v Playwright Inspectorju

Ko končamo z interakcijo s stranjo, pritisnemo gumb Record, ki ustavi snemanje, nato pa z gumbom za kopiranje ustvarjeno kodo prenesemo v odložišče. Sejo brskalnika in inšpektorja prekinemo tako, da zapremo brskalnik, ali pa direktno v terminalu, kjer smo izvedli ukaz [21].

### 3.3 Zvočno testiranje

Zvočno testiranje lahko v Playwrightu izvajamo s pomočjo Chromovih zastavic (angl. flags), ki jih nastavimo kot argumente pri kreiranju seje brskalnika. Ogrodje za testiranje aplikacije z govornim vmesnikom, kjer se od uporabnika

pričakuje vnos skozi mikrofona, si lahko pogledamo na spodnji kodi.

```
import wave
import os
from time import sleep
from playwright.sync_api import Playwright, expect

def test_narekovanje(playwright: Playwright) -> None:
    audio = "\\Audio files\\input.wav"
    audioPath = os.getcwd() + audio
    chromium = playwright.chromium
    browser = chromium.launch(headless=False, args=[
        '--use-fake-device-for-media-stream',
        '--use-fake-ui-for-media-stream',
        '--use-file-for-fake-audio-capture=
        {0}'.format(audioPath)])
    context = browser.new_context()
    context.grant_permissions(permissions=['microphone'])
    page = context.new_page()

    login(page)
    playAudio(page, audioPath)
    # -----
    context.close()
    browser.close()
```

Glavni del kode je spremenljivka `browser`, kjer s posebnimi argumenti uka-  
zne vrstice, vključno z uporabo lažne naprave za pretakanje medijev, lažnega  
uporabniškega vmesnika za pretakanje medijev in uporabo datoteke za lažni  
zajem zvoka, ki je nastavljena na spremenljivko `audioPath`, omogočimo br-  
skalniku predvajanje zvočne datoteke. Za tem se kreira nov kontekst, ki se  
mu dodeli dovoljenje za uporabo mikrofona, nato pa se v kontekstu ustvari

nova stran, preko katere bomo izvajali ukaze, navedene v poglavjih 3.2.1 in 3.2.2.

Predvajanje zvočne datoteke deluje tako, da ko brskalnik uporabnika prosi za dovoljenje za uporabo mikrofona, ki ga v našem primeru avtomatsko potrdim, Chrome začne v zanki predvajati dodeljen zvočni posnetek. Pri specifičnih primerih uporabe želimo testirati določene zvočne ukaze aplikacije, zato moramo posnetek predvajati samo enkrat in preveriti rezultate. To lahko izvedemo tako, da uporabimo knjižnico `wave` v kombinaciji s `sleep`, da za dolžino zvočnega posnetka prekinemo nadaljnje izvajanje programa. Po preteklem času nadaljujemo s testiranjem izgovorjenega besedila ali ostalih funkcionalnosti aplikacije. Kodo za predvajanje posnetka si lahko pogledamo spodaj.

```
page.get_by_role("button", name="Začni predvajati").click()
```

```
with wave.open(path) as mywav:
    frames = mywav.getnframes()
    rate = mywav.getframerate()
    duration = frames / float(rate)
    sleep(duration)
```

```
page.get_by_role("button", name="Končaj").click()
```

Za dosnemavanje ponovno pritisnemo na gumb začni predvajati in program bo začel predvajati zvočni posnetek, naveden v argumentih brskalnika. Težava pri dosnemavanju nastopi, ko želimo uporabiti dve različni vhodni datoteki, ki pa jih trenutno zastavica, uporaba datoteke za lažni zajem zvoka, ne omogoča. S tem nam možnosti testiranja malenkost omeji, vendar se da z avtomatskimi testi vseeno pokrit večina, če ne vse primere uporabe.

Kar se tiče samih zvočnih posnetkov, se priporoča, da je na začetku posnetka nekaj sekund tišine, da se pravilno vzpostavi seja med čelnim in zalednim delom ter da so vsi posnetki v formatu `wav` in ne `mp3` ali katerem od ostalih možnih formatov.

### 3.4 Izvajanje testov

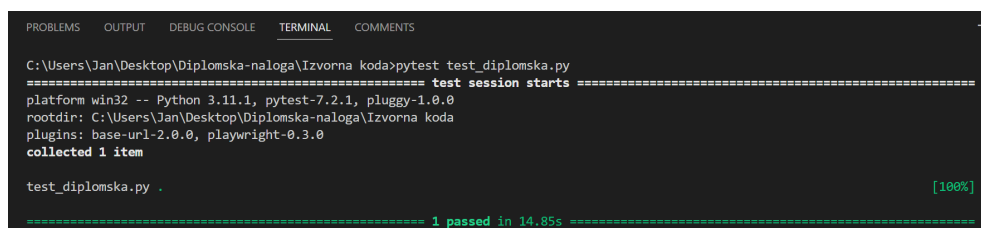
Privzeto se bodo testi izvajali na Chromu. To lahko spremenimo z različnimi CLI možnostmi. Testi se izvajajo v načinu brez glave, kar pomeni, da se med izvajanjem testov ne odpre uporabniški vmesnik brskalnika. Rezultati testov in dnevniki testov bodo prikazani v terminalu. Izvajamo lahko posamezne teste ali vse hkrati. Testi se prav tako lahko izvajajo v enem ali več brskalnikih. Imena datotek se morajo pričeti s `test_`, da jih `pytest` prepozna kot izvedljive datoteke ob uporabi ukaza. Če želimo, da `pytest` izvede samo eno datoteko, uporabimo ukaz:

```
pytest test_test.py
```

ki mu lahko dodamo CLI argumente glede na želene specifikacije. Najbolj pogosto uporabljene možnosti so sledeče [15]:

- **--headed** izvajanje testov v načinu z glavo;
- **--browser** izvajanje testov v drugem brskalniku firefox, webkit;
- **--slowmo** izvajanje testov s počasnim posnetkom;
- **--video** ali želimo posneti videoposnetek za vsak test;
- **--screenshot** ali naj se po vsakem testu samodejno zajame posnetek zaslona.

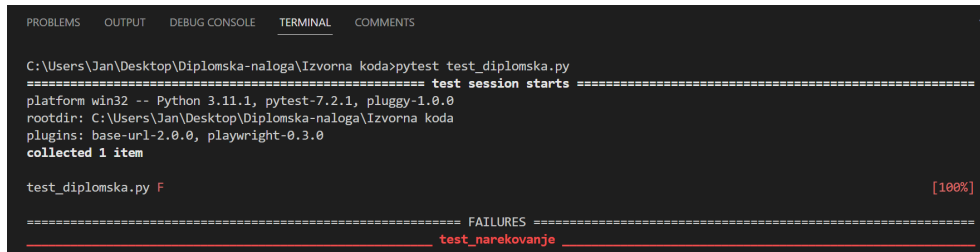
Primer uspešno izvedenega testa lahko vidimo na sliki 3.3, neuspešno izvedenega pa na sliki 3.4.

A screenshot of a terminal window with a dark background. At the top, there are tabs for 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL' (which is active), and 'COMMENTS'. The terminal shows the command `C:\Users\Jan\Desktop\Diplomska-naloga\Izvorna koda>pytest test_diplomska.py`. Below this, it displays the start of a test session with details like platform (win32), Python version (3.11.1), pytest version (7.2.1), and plugins (base-url-2.0.0, playwright-0.3.0). It states 'collected 1 item'. Then, it shows a single test passing: `test_diplomska.py .` with a green progress bar at [100%]. At the bottom, it summarizes: `1 passed in 14.85s`.

Slika 3.3: Primer uspešno izvedenega testa

V primeru, da Playwright elementa ne najde na strani, test avtomatsko prekine po 30000 ms in javi `TimeoutError`. Dnevnik (angl. `log`) nam tudi

pove, v kateri vrstici je program nehal delovati, v primeru funkcije expect pa pričakovan in vrnjen rezultat programa.



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  COMMENTS

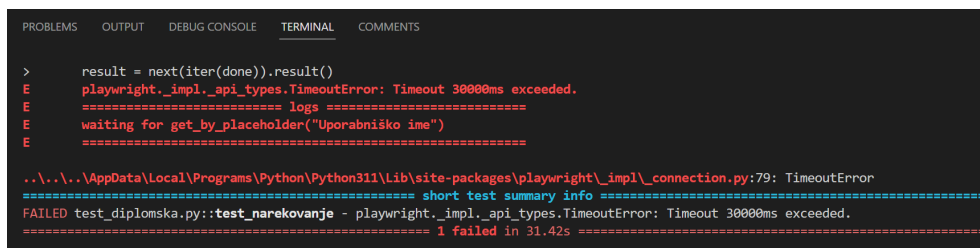
C:\Users\Jan\Desktop\Diplomska-naloga\Izborna koda>pytest test_diplomska.py
===== test session starts =====
platform win32 -- Python 3.11.1, pytest-7.2.1, pluggy-1.0.0
rootdir: C:\Users\Jan\Desktop\Diplomska-naloga\Izborna koda
plugins: base-url-2.0.0, playwright-0.3.0
collected 1 item

test_diplomska.py F [100%]

===== FAILURES =====
test_narekovanje
```

Slika 3.4: Primer neuspešno izvedenega testa

Na sliki 3.5 lahko vidimo razlog za potek časovnega roka (angl. timeout).



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  COMMENTS

> result = next(iter(done)).result()
E playwright._impl._api_types.TimeoutError: Timeout 30000ms exceeded.
E ===== logs =====
E waiting for get_by_placeholder("Uporabniško ime")
E =====

..\..\AppData\Local\Programs\Python\Python311\Lib\site-packages\playwright\_impl\_connection.py:79: TimeoutError
===== short test summary info =====
FAILED test_diplomska.py::test_narekovanje - playwright._impl._api_types.TimeoutError: Timeout 30000ms exceeded.
===== 1 failed in 31.42s =====
```

Slika 3.5: Dnevnik (angl. log) neuspešnega testa

## Poglavje 4

# Namestitev in uporaba Seleniuma

V poglavju bodo predstavljene možnosti uporabe Seleniuma v Pythonu, pisanje testov, testiranje aplikacije z govornim vmesnikom in izvajanje napisanih testov. Opisani bodo odseki kode in najpomembnejše stvari, zato si lahko celotno kodo pogledamo na Githubu [5].

### 4.1 Priprava okolja

Selenium, tako kot Playwright, omogoča pisanje testov v več programskih jezikih in zaradi starosti ter uveljavljenosti v svetu avtomatskega testiranja podpira, poleg glavnih jezikov, kot so Python, Javascript in C# tudi Ruby, Perl in PHP. Vseeno pa bomo tudi za Selenium uporabili Python, predvsem zaradi enostavnosti, saj ne zahteva naprednih nastavitev kot drugi jeziki.

Za pripravo okolja za testiranje govornega vmesnika je potrebno na sistemu najprej namestiti Python 3 in preko pip namestiti Selenium knjižnico. Poleg tega bomo uporabili tudi knjižnice wave, os in time, ki nam bodo v pomoč pri delu z zvočnimi datotekami v različnih formatih ter upravljanju s časom in operacijskim sistemom. Na primer, knjižnica wave nam nudi enostaven vmesnik za izračun dolžine zvočnega posnetka med predvajanjem, medtem

ko funkcija `sleep()` iz knjižnice `time` omogoča ustavitev izvajanja programa za določeno število sekund. Z uporabo knjižnice `os` pa lahko sestavimo poti do zvočnih posnetkov.

Za izvedbo testov bomo uporabili tudi Chromium WebRTC zastavice, ki omogočajo direktno izmenjavo avdio, video in podatkov med spletnimi brskalniki ter drugimi aplikacijami za medsebojno povezovanje. WebRTC je tehnologija, vgrajena v večino sodobnih spletnih brskalnikov, ki uporablja nabor tehnologij, vključno z zvočnimi in video kodeki, šifriranjem in omrežnimi protokoli, ki zagotavljajo varno in zanesljivo komunikacijo. S pomočjo WebRTC zastavic lahko učinkovito izvajamo teste za preverjanje delovanja spletnih aplikacij, ki uporabljajo govorni vmesnik.

Selenium za razliko od Playwrighta ne čaka avtomatsko, da se elementi in stran naloži, zato treba med ukazi nastaviti primeren časovni razpon s funkcijo `sleep()`. Prav tako moramo pri Seleniumu ročno namestiti gonilnik za brskalnik, ki ga pri Playwrightu dobimo z ukazom. Ker bomo uporabljali Chromium, najprej iz uradne spletne strani [2] prenesemo najnovejšo različico Chromedriverja oz. isto različico, kot je naš Chrome brskalnik. Postopek za namestitve ostalih brskalnikov je podoben. Ko imamo na računalniku `chromedriver.exe`, ga prestavimo v direktorij, kjer se bo nahajal naš Python Selenium test. Točni ukazi in postopek namestitve so na voljo na Github repozitoriju [5].

## 4.2 Pisanje testov

Pisanje Selenium testov je zelo podobno Playwrightu z dvema ključnima razlikama. Playwright ima vgrajeno funkcijo samodejnega čakanja, kar pomeni, da po vsaki akciji počaka, da se stran in elementi naložijo pred izvedbo akcije nad njimi, Selenium pa te funkcionalnosti nima in moramo zato uporabiti funkcijo `sleep()` ali teste pisati asinhrono. Za namene naloge smo teste implementirali sinhrono in med ukazi program prekinili za sekundo. Druga razlika pa je, da Selenium ne omogoča avtomatskega generiranja testov in moramo



vsak test napisati ročno, kar nam oteži in podaljša delo.

### 4.2.1 Locator

Selenium iskalniki oz. lokatorji se uporabljajo za iskanje spletnih elementov med avtomatizacijo testiranja. Ti lokatorji omogočajo, da Selenium komunicira z različnimi vrstami spletnih elementov, kot so besedilna polja, gumbi in povezave. Najpogosteje uporabljeni lokatorji, ki jih ponuja Selenium, so sledeči: [18]

- **class name** poišče elemente, katerih ime razreda vsebuje iskano vrednost (sestavljena imena razredov niso dovoljena);
- **css selector** poišče elemente, ki ustrezajo selektorju CSS;
- **id** poišče elemente, katerih atribut ID se ujema z iskalno vrednostjo;
- **name** poišče elemente, katerih atribut NAME ustreza iskani vrednosti;
- **link text** poišče elemente sidra, katerih vidno besedilo se ujema z iskalno vrednostjo;
- **partial link text** poišče elemente sidra, katerih vidno besedilo vsebuje iskano vrednost. Če se ujema več elementov, bo izbran le prvi;
- **tag name** poišče elemente, katerih ime oznake se ujema z iskalno vrednostjo;
- **xpath** poišče elemente, ki ustrezajo izrazu XPath.

Ko uporabljamo Selenium za avtomatizacijo testiranja spletne strani, je ena od ključnih funkcij interakcija z elementi na strani. Ko najdemo element, ki ga želimo nadzorovati, imamo na voljo številne metode, ki jih lahko uporabimo za interakcijo z elementom in preverjanje njegovega vedenja. Med najpogosteje uporabljenimi metodami so spodaj našteje funkcije: [17]

- **send\_keys()** Vnese besedilo v besedilno polje ali tekstovno polje;
- **click()** klikne na element, kot je gumb ali povezava;
- **submit()** odda obrazec, če je element obrazec ali gumb;
- **clear()** izbriše besedilo iz besedilnega polja ali besedilne površine;
- **get\_attribute()** pridobi vrednost določenega atributa elementa;
- **text()** pridobi vidno notranje besedilo elementa;
- **is\_displayed()** preveri, ali je element viden na strani;
- **is\_enabled()** preveri, ali je element omogočen;
- **is\_selected()** preveri, ali je izbirno polje ali radijski gumb izbran.

#### 4.2.2 Trditve

Selenium trditve (angl. assertions) se uporabljajo pri avtomatskem testiranju za preverjanje, ali je določen pogoj izpolnjen ali ne. Trditve pomagajo zagotoviti, da testirana aplikacija ustvari pričakovani rezultat. Selenium ponuja več vrst trditev, kot so: [16]

- **AssertEquals** trditev primerja dejanske in pričakovane vrednosti in je uspešna, če sta enaki;
- **AssertNotEquals** trditev je uspešna, če dejanska in pričakovana vrednost nista enaki;
- **AssertTrue** trditev je uspešna, če je pogoj, ki ji je posredovan, resničen;
- **AssertFalse** trditev je uspešna, če je pogoj, ki ji je posredovan, napačen;
- **AssertNull** trditev je uspešna, če je določen predmet null;
- **AssertNotNull** trditev je uspešna, če navedeni predmet ni null.

Trde trditve	Mehke trditve
Izvajanje testa se prekine, če pogoj trditve ni izpolnjen.	Izvajanje testa se bo nadaljevalo do konca testnega primera, tudi če pogoj trditve ni izpolnjen.
Za zajemanje trditev ni treba klicati nobenih metod.	Za prikaz rezultatov trditev na koncu testa mora tester priklicati <code>assertAll()</code> funkcijo.

Tabela 4.1: Vrste trditev

Selenium trditve so bistven del pisanja robustnih in zanesljivih avtomatskih testov. Pomagajo zagotoviti, da aplikacija deluje pravilno ter da so testi sami natančni in zanesljivi. Same trditve pa se delijo na dva tipa, trde in mehke trditve. Primerjavo lahko vidimo v tabeli 4.1 [23].

### 4.2.3 Enostaven primer Selenium testa

Primer Selenium testa, ki uporabi našteje funkcionalnosti, lahko vidimo v spodnji psevdokodi. Prikazan primer se z elektronskim naslovom in geslom prijavi v Gmail in z uporabo trditve preveri, če je naslov strani res Gmail.

```
from selenium import webdriver
from selenium.webdriver.common.by import By

driver = webdriver.Chrome()
driver.get("https://accounts.google.com/")
driver.find_element(By.NAME, "email").send_keys("email")
driver.find_element(By.ID, "password").send_keys("password")
assert driver.title == "Gmail"
driver.quit()
```

## 4.3 Zvočno testiranje

Zvočno testiranje lahko s Seleniumom izvajamo na podoben način kot pri Playwrightu. Enako bomo uporabili tudi Chromove zastavice, ki jih uporabimo tako, da simuliramo virtualni mikrofonski, skozi katerega bomo brskalniku in posledično aplikaciji predvajali zvočno datoteko. Ograjeno za testiranje aplikacije z govornim vmesnikom, kjer uporabnik govori v mikrofonski, lahko vidimo na spodnji kodi.

```
import wave
import os
from time import sleep
from selenium import webdriver
from selenium.webdriver.chrome.options import Options
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By

def test_audio():
    opt = Options()
    opt.add_argument("--use-fake-device-for-media-stream")
    opt.add_argument("--use-fake-ui-for-media-stream")
    opt.add_argument("--use-file-for-fake-audio-capture="
                    "{0}".format(audioPath))
    service = Service(
        executable_path="{0}".format(driverPath))
    driver = webdriver.Chrome(service=service, options=opt)

    login(driver, aplikacijaUrl)
    playAudio(driver, audioPath)
    driver.quit()
```

Najpomembnejši del kode so opcije, kjer nastavimo Chromove zastavice in s tem omogočimo programu simuliranje virtualnega mikrofona, skozi katerega predvajamo našo zvočno datoteko. Enako kot pri Playwrightu uporabimo zastavice za uporabo lažne naprave za pretakanje medijev, lažnega uporabniškega vmesnika za pretakanje medijev in uporabo datoteke za lažni zajem zvoka, ki je nastavljena na spremenljivko `audioPath`. Drugi pomemben del pa je spremenljivka `service`, kjer programu podamo pot do gonilnika za uporabljen brskalnik. Obe spremenljivki nato skupaj uporabimo pri inicializaciji instance razreda `Chrome()`, ki ga nato uporabimo za izvajanje testov. Tako kot pri Playwrightu moramo predvajanje zvočne datoteke nekoliko prilagoditi, saj ga `Chrome` izvaja v zanki. Enako kot v prejšnjem poglavju uporabimo knjižnico `wave` v kombinaciji s funkcijo `sleep()`, da za dolžino posnetka prekinemo nadaljnje izvajanje programa.

Tudi Selenium tako kot Playwright ne omogoča predvajanje dveh različnih zvočnih datotek, zato se moramo pri testiranju več funkcionalnosti znajti, ali pa problem razbiti na več manjših in testirati vsako komponento posebej. Ista pravila glede zvočnega posnetka veljajo tudi pri Seleniumu, saj je priporočljivo na začetku posnetka pustiti nekaj sekund tišine, da se v celoti vzpostavi zveza med čelnim in zalednim delom ter simulatorjem mikrofona.

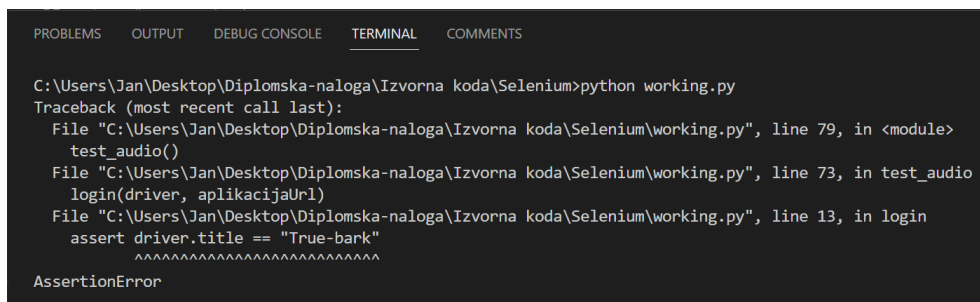
## 4.4 Izvajanje testov

Za razliko od Playwrighta, kjer lahko v kodi ali v ukazni vrstici določimo razne parametre za izvedbo testa, Selenium vse prejme preko argumentov neposredno v kodi. Tako kot smo dodali zastavice za simuliranje mikrofona, brskalniku povemo, če želimo, da se izvajanje začne v načinu brez beleženja zgodovine, v celozaslonskem načinu ali izberemo eno izmed številnih opcij, ki jih ponuja Selenium: [1]

- **start-maximized** Chrome se odpre v celozaslonskem načinu;
- **incognito** Chrome se odpre v načinu brez beleženja zgodovine;

- **headless** Chrome se odpre v načinu brez uporabniškega vmesnika tako, da ne vidimo izvajanje testa;
- **disable-extensions** onemogoči obstoječe razširitve v brskalniku Chrome;
- **disable-infobars** prepreči, da bi Chrome prikazal obvestilo "Chrome is being controlled by automated software".

Teste izvedemo tako kot navadno Python datoteko. V primeru uspešno izvedenega testa nam Selenium v ukazni vrstici ne javi ničesar, javi pa napako v primeru napačne trditve, ki si jo lahko pogledamo na sliki 4.1. Vseeno nam ne da brez dodatnih knjižnic bolj podrobne razlage, zakaj se test ni izvedel oz. nam ne izpiše dnevnika (angl. log) uspešno izvedenih testov.



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  COMMENTS

C:\Users\Jan\Desktop\Diplomska-naloga\Izvorna koda\Selenium>python working.py
Traceback (most recent call last):
  File "C:\Users\Jan\Desktop\Diplomska-naloga\Izvorna koda\Selenium\working.py", line 79, in <module>
    test_audio()
  File "C:\Users\Jan\Desktop\Diplomska-naloga\Izvorna koda\Selenium\working.py", line 73, in test_audio
    login(driver, aplikacijaUrl)
  File "C:\Users\Jan\Desktop\Diplomska-naloga\Izvorna koda\Selenium\working.py", line 13, in login
    assert driver.title == "True-bark"
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
AssertionError
```

Slika 4.1: Primer neuspešno izvedenega Selenium testa

## Poglavje 5

### Sklepne ugotovitve

Rezultate orodij, ki smo ju posamezno obdelali v prejšnjih poglavjih, lahko sedaj povežemo v celoto, kjer izpostavimo ključne ugotovitve. Tekom diplomske naloge smo analizirali orodja za kreiranje avtomatskih testov za odjemalce z govornim vmesnikom. V Poglavju 2 smo izmed mnogih izbrali dve najbolj primerni za nadaljnje teste in uporabo v praksi. Po kratki predstavitvi uporabljenih orodij smo se lotili praktičnega dela naloge, kjer smo v dveh poglavjih predstavili orodji Playwright in Selenium. Za obe smo opisali pripravo okolja za testiranje, pisanje avtomatskih testov in funkcionalnosti, ki jih vsako od orodij ponuja, opisali in predstavili zvočno testiranje, ki predstavlja jedro naše diplomske naloge, in na koncu pokazali izvajanje napisanih skript.

Naši rezultati kažejo, da sta orodji Selenium in Playwright podobni v funkcionalnostih, ki jih ponujata za avtomatsko testiranje spletnih aplikacij, vendar pa obstajajo določene razlike, ki jih je vredno izpostaviti. V nadaljevanju je tabela 5.1 z naborom funkcionalnosti in primerjalna analiza. Kot je razvidno iz tabele, sta orodji Selenium in Playwright zelo podobni v funkcionalnostih, ki jih ponujata, kot so podpora za več brskalnikov, podpora za več programskih jezikov, možnost upravljanja z zavihki, simulacija klikov in tipkanja ter paralelno testiranje. Vendar pa obstajajo nekatere razlike v načinu, kako orodji izvajata nekatere funkcionalnosti.

<b>Funkcionalnost</b>	<b>Selenium</b>	<b>Playwright</b>
Podpora za več brskalnikov	✓	✓
Podpora za več programskih jezikov	✓	✓
Možnost upravljanja z zavihki	✓	✓
Podpora za dinamično čakanje elementov	✗	✓
Podpora za simulacijo klikov in tipkanja	✓	✓
Snemanje testov	Omejeno	✓
Zvočno testiranje	✓	✓
Paralelno testiranje	✓	✓
Enostavnost namestitve in uporabe	Zahtevno	Enostavno

Tabela 5.1: Primerjava funkcionalnosti med Playwright in Selenium.

Največja razlika med orodjema Selenium in Playwright je v enostavnosti namestitve in uporabe. Medtem ko je namestitev in uporaba Playwrighta preprosta, zahteva Selenium namestitev gonilnikov in drugih dodatkov za uporabo, kar lahko oteži začetke uporabe orodja. Poleg tega Playwright ponuja tudi funkcionalnost snemanja testov, kar omogoča hitrejše in lažje pisanje avtomatskih testov. V primerjavi s tem je funkcionalnost snemanja testov v Seleniumu omejena.

V sklopu diplomske naloge smo uspešno implementirali avtomatsko testiranje za odjemalce z govornim vmesnikom za obe orodji, Selenium in Playwright. Za to smo uporabili funkcionalnosti, ki jih ponujata ti dve orodji, ter izdelali primerne teste za preverjanje delovanja odjemalcev. Pri tem smo ugotovili, da je Playwright boljše orodje od Seleniuma, saj ima več funkcionalnosti, ki nam omogočajo lažjo in boljšo izvedbo testov. Ob tem smo dobili tudi vpogled v delovanje obeh orodij ter spoznali, da obstajajo tudi druga orodja, ki bi jih lahko uporabili za avtomatsko testiranje spletne aplikacije. Vendar smo se osredotočili na omenjeni dve, saj smo želeli primerjati njune zmogljivosti in ugotoviti, katero je bolj primerno za testiranje odjemalcev z govornim vmesnikom.



Kljub uspešni izvedbi testov pa smo ugotovili, da ima avtomatsko testiranje z zvočnim vmesnikom še vedno nekatere omejitve. Ena izmed njih je ta, da lahko uporabimo le en zvočni posnetek na test, kar lahko malo omeji možnosti testiranja. Zato predlagamo nadgradnjo, ki bi omogočila izvedbo več testov hkrati in tako omogočila bolj celovito testiranje odjemalcev. Ena izmed možnih nadgradenj je uporaba operacij na nivoju operacijskega sistema, ki bi omogočile spreminjanje izhoda zvočnika v vhod mikrofona.

Skupno gledano smo s to diplomsko nalogo uspešno izvedli avtomatsko testiranje za odjemalce z govorim vmesnikom ter ugotovili, da je Playwright boljše orodje od Seleniuma za izvedbo tovrstnih testov. Hkrati smo spoznali, da obstajajo še druga orodja, ki bi jih lahko uporabili za avtomatsko testiranje spletne aplikacije ter predlagali nadgradnjo testiranja za bolj celovito preverjanje delovanja odjemalcev.



# Celotna literatura

- [1] *Chromove opcije za testiranje*. URL: <https://www.guru99.com/chrome-options-desiredcapabilities.html> (pridobljeno 5. 3. 2023).
- [2] *Chromedriver uradna spletna stran*. URL: <https://chromedriver.chromium.org/downloads> (pridobljeno 4. 3. 2023).
- [3] *Cypress dokumentacija*. URL: <https://docs.cypress.io/guides/overview/why-cypress> (pridobljeno 12. 3. 2023).
- [4] Boni Garcia in sod. "Testing framework for WebRTC services". V: *Proceedings of the 9th EAI International Conference on Mobile Multimedia Communications*. 2016, str. 40–47.
- [5] *Repozitorij z kodo in navodili*. URL: <https://github.com/FakeJan/Diplomska-naloga> (pridobljeno 12. 3. 2023).
- [6] Waweru Mwaura. *End-to-End Web Testing with Cypress: Explore techniques for automated frontend web testing with Cypress and JavaScript*. Packt Publishing Ltd, 2021.
- [7] Charul Nigam in Sushma Malik. "Comparative Analysis of Automation Testing Tools". V: *IITM Journal of Information Technology* 45 (2018).

- 
- [8] *Python os knjižnica*. URL: <https://docs.python.org/3/library/os.html> (pridobljeno 12. 2. 2023).
- [9] *Playwright trditve*. URL: <https://playwright.dev/python/docs/test-assertions> (pridobljeno 19. 2. 2023).
- [10] *Playwright Python dokumentacija*. URL: <https://playwright.dev/python/docs/intro> (pridobljeno 11. 2. 2023).
- [11] *Playwright lokatorji*. URL: <https://playwright.dev/python/docs/locators> (pridobljeno 19. 2. 2023).
- [12] *Playwright spletna stran*. URL: <https://playwright.dev/> (pridobljeno 12. 3. 2023).
- [13] Abdelmalek Rahal. "Concept and Implementation of an automated GUI-Test Suite based on Squish Framework". Doktorska disertacija. Ulm University, 2021.
- [14] *Robot Framework spletna stran*. URL: <https://robotframework.org/> (pridobljeno 12. 3. 2023).
- [15] *Izvajanje Playwright testov s pytest*. URL: <https://playwright.dev/python/docs/running-tests> (pridobljeno 25. 2. 2023).
- [16] *Selenium trditve*. URL: <https://www.javatpoint.com/selenium-assertions> (pridobljeno 5. 3. 2023).
- [17] *Selenium interakcije z elementi*. URL: <https://www.selenium.dev/documentation/webdriver/elements/interactions/> (pridobljeno 5. 3. 2023).

- 
- [18] *Selenium lokatorji*. URL: <https://www.selenium.dev/documentation/webdriver/elements/locators/> (pridobljeno 5.3.2023).
- [19] Sherolwendy Anak Sualim, Noraniah Mohd Yassin in Radziah Mohamad. "Comparative evaluation of automated user acceptance testing tool for web based application". V: *International Journal of Software Engineering and Technology* 2.2 (2017).
- [20] AT Sudhan, G Paramesh in G Ranjani. "Automation and Integration of SSI Test Cases for Abis and A-Interface in GSM Using Robot Framework". V: *Computer Networks and Inventive Communication Technologies: Proceedings of Third ICCNCT 2020*. Springer. 2021, str. 837–851.
- [21] *Automatsko generiranje testov*. URL: <https://playwright.dev/python/docs/codegen-intro> (pridobljeno 25.2.2023).
- [22] *Python time knjižnica*. URL: <https://docs.python.org/3/library/time.html> (pridobljeno 12.2.2023).
- [23] *Selenium trde in mehke trditve*. URL: <https://www.browserstack.com/guide/verify-and-assert-in-selenium> (pridobljeno 5.3.2023).
- [24] Nishant Verma. *Mobile Test Automation With Appium*. Packt Publishing Ltd, 2017.
- [25] *Python wave knjižnica*. URL: <https://docs.python.org/3/library/wave.html> (pridobljeno 12.2.2023).
- [26] Dongsong Zhang in Boonlit Adipat. "Challenges, methodologies, and issues in the usability testing of mobile applications". V: *International journal of human-computer interaction* 18.3 (2005), str. 293–308.

- [27] Dmitry Zhyhulin, Kostiantyn Kasian in Mykola Kasian. “Combined method of prioritization and automation of software regression testing”. V: *2022 IEEE 16th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET)*. IEEE. 2022, str. 751–755.