

UNIVERSITATEA BABEŞ-BOLYAI CLUJ-NAPOCA
FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ
SPECIALIZAREA INFORMATICĂ

LUCRARE DE LICENȚĂ

Testarea Interfețelor Grafice

Conducător științific
Prof. Dr. Czibula Istvan

Student
Olaru Laura-Elena

2022

Abstract

This thesis is about the study in the field of graphical user interface testing, discovering different perspectives in the field, such as the importance of testing, what each type of testing means, what are the advantages and disadvantages of each type of testing, what concepts and methodologies are needed to study and what frameworks and tools are needed for each type of testing. The types of testing studied are manual testing, automated testing using scripts, automated testing, testing using Record and Playback, and testing using Robotic Process Automation. Regarding manual testing, we discovered different testing methodologies that consist of techniques based on the experience of testers and techniques based on the states of a finite state machine built following the interaction with the graphical user interface. We found out how to automate scripting testing and what are the different methodologies for testing scripts, after the graphical interface changes. We also noticed the impact of changing the graphical user interface to the different types of desktop, web and mobile interfaces. We went through the different frameworks and testing technologies using scripts, with their evolution over time and what features each one of them has.

Automatic testing has many advantages over manual testing. Along with the different features that different interfaces have, it is also possible to identify what kind of tools and frameworks are appropriate for testing depending on the case. On the other hand, testing using Record and playback improves automatic testing by recording actions on the graphical interface and repeating them. Thus, We studied what this mechanism consists of and how the frameworks brought improvements in the process of testing the graphical interface. Also, testing using Robotic Process Automation (RPA) brings new features to facilitate the testing process. We studied what testing using software robots is and how it influences people's productivity compared to other types of testing. Finally, we concluded what were the difficulties in testing graphical user interfaces and how they differ comparatively, as well as which could be the future directions for developing the graphical user interfaces testing.

This thesis is structured in 4 chapters, where the first 2 chapters consists of the introduction and the theoretical research and foundation of the study of graphical user interface testing and the last 2 chapters consist of the implemented application that was tested, along with the conclusions.

Cuprins

1	Introducere	1
2	Testarea interfețelor grafice	3
2.1	Testarea manuală	4
2.1.1	Avantaje și dezavantaje ale testării manuale	4
2.1.2	Concepțe și metodologii de bază	5
2.2	Automatizarea testării folosind scripturi	7
2.2.1	Avantaje și dezavantaje ale testării folosind scripturi	7
2.2.2	Interfete desktop, web și mobil	7
2.2.3	Concepțe și metodologii de bază	8
2.2.4	Framework-uri existente	10
2.3	Testarea automată	14
2.3.1	Avantaje și dezavantaje ale testării automate	14
2.3.2	Interfete desktop, web și mobil	15
2.3.3	Concepțe și metodologii de bază	16
2.3.4	Framework-uri existente	17
2.4	Testarea folosind Record and Playback	19
2.4.1	Avantaje și dezavantaje ale testării Record and Playback . . .	19
2.4.2	Concepțe și metodologii de bază	20
2.4.3	Framework-uri existente	21
2.5	Testarea folosind Robotic Process Automation	23
2.5.1	Avantaje și dezavantaje ale testării Robotic Process Automation	23
2.5.2	Framework-uri existente	24
2.5.3	Concepțe și metodologii de bază	25
2.6	Dificultăți în testarea interfețelor grafice	28
3	Proiect: Aplicație de postare de proiecte	30
3.1	Analiza aplicației	30
3.1.1	Descrierea aplicației	30
3.1.2	Functionalitățile aplicației	30
3.1.3	Manual de utilizare	31

3.2	Proiectarea aplicatiei	36
3.2.1	Arhitectura client server	36
3.2.2	Persistența datelor în baza de date	38
3.3	Implementarea aplicației	39
3.3.1	Arhitectura claselor aplicației	39
3.3.2	Tehnologii folosite în implementarea aplicației	40
3.3.3	Arhitectura interfețelor web și mobil	44
3.4	Testarea aplicației	45
3.4.1	Framework-uri folosite în testarea interfeței grafice	45
3.5	Comparatii cu aplicații similare	48
4	Concluzii	50
Bibliografie		52

1. Introducere

Principalul obiectiv al acestei lucrări este de evidenția importanța testării interfeței grafice a unei aplicații înainte ca aceasta să intre în producție. Din moment ce aproape orice aplicație ce ajunge în producție prezintă o interfață grafică ce poate fi folosită de către utilizatori, este important ca interfața grafică să respecte standardele din producție și să funcționeze corespunzător scopului în care a fost implementată, fără ca utilizatorii să întâmpine probleme atunci când folosesc aplicația. Pentru ca acest aspect să fie îndeplinit, există diferite moduri prin care un programator poate testa interfața grafică și există multe instrumente de testare care se concentrează pe diferite aspecte ale acesteia, motiv pentru care pe baza unei aplicații implementate de mine explorez diferitele moduri de testare a interfeței grafice.

Capitolele teoretice din această lucrare se concentrează pe aceste tehnici de testare a interfeței grafice, în ce constau acestea, care sunt avantajele și dezavantajele lor, conceptele și metodologiile folosite, impactul lor asupra diferitelor interfețe desktop, web și mobil și framework-urile existente pentru fiecare tip de testare. Printre acestea se numără testarea manuală, automatizarea testării folosind scripturi, testarea automată, testarea record and playback și testarea folosind Robotic Process Automation (RPA). În cele din urmă concluzionez care sunt dificultățile întâlnite în testarea interfeței grafice și viitoarele posibile continuări. Testarea manuală se referă la testarea graduală prin parcurgerea elementelor interfeței grafice și verificarea funcționării corecte ale acestora de către tester. Testarea bazată pe scripturi automatizează testarea prin execuția de acțiuni determinate de scripturi asupra elementelor interfeței grafice. Testarea automată are la bază automatizarea procesului de testare prin folosirea de instrumente specifice. Testarea record and playback se concentrează pe înregistrarea acțiunilor utilizatorului și reproducerea lor sub formă de testare. Testarea RPA se referă la folosirea robotilor pentru identificarea și parcugerea automată a proceselor ce constau în testarea interfeței grafice. Prin parcugerea acestor metode de testare se are în vedere atât funcționarea lor pe diferite tipuri de interfețe, precum desktop, web și mobil, cât și framework-urile existente ce pot fi folosite în testare.

Am dezvoltat aplicația Projy, ce are drept scop postarea proiectelor personale de către utilizatori din toate domeniile. Cu ajutorul ei se testează în diferite mo-

duri interfață grafică pe mai multe tipuri de dispozitive: IOS, Android și Web. De asemenea, aceste interfețe grafice sunt formate din mai multe componente: ecrane diferite, împreună cu navigarea dintre ele, diferitele funcționalități care trebuie să funcționeze conform așteptărilor și funcționarea corectă a widget-urilor aplicației, împreună cu respectarea standardelor de interfață grafică din producție.

Capitolele ce au în vedere această aplicație se concentrează pe analiza funcționalităților aplicației, detaliile de proiectare la nivel de arhitectură, tehnologiile folosite la nivel de implementare, testarea aplicației, partea de deployment și în final comparații cu aplicații similare. În ceea ce privește analiza funcționalităților aplicației, se urmăresc scopul aplicației, diagramele corespunzătoare și manualul de utilizare. Detaliile de proiectare la nivel de arhitectură au în vedere ansamblul aplicației, conectarea dintre client și server, persisența datelor și schița arhitecturii. Implementarea aplicației se axează pe framework-urile și tehnologiile folosite, iar testarea aplicației are în vedere unit și integration testing. Partea de deployment ilustrează pașii de punere a aplicației pe server. În ceea ce privește compararea cu aplicații similare, se face o paralelă dintre aplicația Projy și ceea ce deja există în industrie.

În final, concluzionez de ce este importantă testarea interfeței grafice a unei aplicații înainte ca aceasta să intre în producție, ce am învățat în urma realizării lucrării și care ar putea fi posibilele continuări și îmbunătățiri în ceea ce privește testarea interfeței grafice.

2. Testarea interfețelor grafice

Aplicațiile cu interfețe grafice se regăsesc în aproape toate domeniile importante din societatea modernă. De exemplu, sistemele bancare folosesc aplicații pentru realizarea de diferite operații bancare precum tranzacții, crearea de depozite sau diferite transferuri bancare importante. În domeniul transporturilor mașinile și avioanele folosesc calculatoare de bord ce sunt gestionate prin intermediul interfețelor grafice și pentru rezervarea curselor sau zborurilor și verificarea orarelor acestora. În domeniul de marketing, oamenii cumpără produse online și le aleg dintr-un catalog virtual. În domeniul educațional, se folosesc platforme de învățare online ce nu ar putea fi folosite fără o interfață grafică funcțională. De asemenea, în domeniul medical sunt folosite diferite aplicații ce ajută la determinarea de simptome ale pacienților, înregistrarea fișelor medicale pentru aceștia, sau chiar și reținerea stocului de medicamente existente în farmacie. Toate aceste domenii esențiale și multe altele depind de interfețe grafice funcționale și intuitive. Orice mică eroare în interfață grafică a unei aplicații poate cauza consecințe uriașe în viața oamenilor care le folosesc. Există exemple de aplicații care nu au fost testate corect și au intrat în producție, cauzând daune uriașe, precum avioane prăbușite, transferuri bancare greșite sau chiar și probleme medicale grave cauzate de erori ale aplicațiilor ce nu au fost testate. Toate acestea demonstrează cât de importantă este testarea interfețelor grafice ale aplicațiilor înainte ca acestea să intre în producție.

Interfețele grafice sunt tipuri de interfețe ce le permit utilizatorilor să interacționeze în diferite moduri cu aplicațiile software, prin elemente grafice și indicatori vizuali ce prezintă informațiile și acțiunile posibile pe care le pot face utilizatorii. Interacțiunea directă cu aceste elemente este cea mai facilă și productivă atunci când sunt luate în considerare nevoile și așteptările utilizatorilor atât din punct de vedere vizual, cât și din punct de vedere al performanței și al funcționării corecte. Precum subliniază [Yan11], utilizatorii nu ar trebui să aibă nevoie de un ghid înainte să folosească aplicația, ci ar trebui să își dea seama intuitiv din modul în care este structurată interfața grafică cu ce elemente ale interfeței trebuie să interacționeze pentru a ajunge la scopul dorit. În cele mai multe cazuri, interfețele grafice sunt scrise într-un stil bazat pe event-uri, în sensul în care acestea suportă diferite acțiuni ale utilizatorilor, precum selectarea de elemente din meniu pentru a ajunge la anumite pagini

ale aplicației, inserarea de text ce urmează a fi procesat sau actualizarea interfeței prin anumite acțiuni determinate de utilizator.

În ceea ce privește testarea acestor aplicații cu interfață grafică, testarea exhaustivă nu este posibilă, deoarece aşa cum evidențiază [KFN99], input-ul programului este prea vast, există mult prea multe path-uri posibile de testare, și design-ul aplicației este foarte dificil de testat. De exemplu, toate widget-urile unei interfețe grafice, cum ar fi butoane, text, elemente de meniu, text field-uri, check box-uri sau butoane radio sunt expuse în același timp utilizatorului, iar un utilizator poate parcurge acele elemente în orice ordine și aceasta ar crea în final un număr de permutări de evenimente extrem de mare, ceea ce face imposibil de parcurs toate cazurile posibile numai pe o singură pagină, cu atât mai mult pe toate paginile aplicației [Yan11]. Cu toate acestea, este important ca măcar o parte din cele mai utilizate secvențe de evenimente din aplicație să fie testate pentru a fi o aplicație sigură de a intra în producție. Pentru a se determina ce urmează a fi testat, se proiectează anumite cazuri de testare. De asemenea, nu doar secvențele sunt cele ce trebuie testate, dar și elementele grafice în sine ale interfeței. Pentru aceasta există mai multe tipuri de testare și anume unit testing, widget testing și integration testing. În domeniul interfețelor grafice, unit testing se referă la testarea comportamentului funcțiilor din spatele elementelor grafice, widget testing înseamnă testarea widget-urilor, adică a elementelor grafice în sine, iar integration testing semnifică testarea flow-ului aplicației, mai precis a secvenței de evenimente pe care le poate parcurge un utilizator. În același timp, este importantă observarea comportamentului aplicației testate pe diferite tipuri de dispozitive, și folosind diferite framework-uri de testare. Apoi, testarea se mai poate structura în modalitățile în care poate fi testată interfața grafică a unei aplicații, precum testarea manuală, testarea bazată pe scripturi, testarea automată, testarea record and playback și testarea folosind Robotic Process Automation (RPA), ce sunt discutate în cele ce urmează.

2.1 Testarea manuală

Testarea manuală constă în testarea graduală prin parcurgerea elementelor interfeței grafice și verificarea funcționării corecte ale acestora de către tester. Aceasta implică construirea de cazuri de testare în urma analizei codului sursă și a specificațiilor.

2.1.1 Avantaje și dezavantaje ale testării manuale

În comparație cu alte metode existente de testare, această metodă nu este la fel de practică deoarece necesită multă muncă repetitivă din partea testerilor. Cu toate acestea, bug-urile și erorile găsite în urma testării manuale pot sugera prezența al-

tor probleme similare în cod și implicit cazurile de testare pot fi adaptate spre a găsi probleme similare în timpul testării [Yan11]. Studii anterioare [JMV04] au arătat faptul că abilitățile testerilor au la fel de mult efect asupra rezultatelor testării precum au tehnicele de proiectare a cazurilor de testare. Drept urmare, experiența testerilor poate fi un factor foarte important în testarea manuală corectă a interfeței grafice a unei aplicații. Această experiență se referă la o logică sau un model mintal ce este format în urma încercării în trecut a altor aplicații cu funcționalități similare. S-a demonstrat faptul că testarea manuală a interfețelor grafice poate fi un avantaj dacă este folosit acest model mintal [KM15], deoarece are potențialul de a acumula o înțelegere unică asupra capacitaților aplicației, comparativ cu metodele formale de testare.

2.1.2 Concepțe și metodologii de bază

Un studiu [IML09] a găsit, în urma observării metodei de lucru a testerilor din mai multe companii, ce practici sunt folosite în testare în funcție de această experiență. S-au identificat două categorii și anume strategiile sesiunilor de testare și tehnicele de executare a testelor. În ceea ce privește strategiile sesiunilor de testare, s-au identificat strategii precum Explorarea interfeței grafice, ce constă în acoperirea tuturor widget-urilor, dar testarea lor se face folosind tehnici bazate pe experiență, sau Explorarea de zone vulnerabile mai este o strategie ce se bazează pe cunoștințele experimentale ale testerului în ceea ce privește identificarea acestor tipuri de zone. În principal, strategiile folosite au în vedere abordarea intuitivă și bazată pe experiență.

Pe de altă parte, tehnicele de executare a testelor au în vedere explorarea funcțiilor izolate, în sensul în care se testează cazurile tipice, unde defectele sunt întâlnite de obicei în testare. De exemplu, Simularea de situații anormale și extreme este o tehnică ce are drept scop evaluarea modului în care o funcționalitate se comportă în afara limitelor și regulilor, sau Comparare în cadrul software este o tehnică ce este folosită în compararea de caracteristici similare în locuri diferite ale aceluiași sistem cu scopul verificării consistenței funcționalităților din cadrul aplicației. Toate acestea sunt practici bazate pe explorare și comparație, dar mai există strategii bazate pe documentație, sau pe datele de input, unde testele se fac pe baza valorilor date și combinarea lor. Această categorie bazată pe date de input este similară cu folosirea de clase de echivalentă, unde datele de input fac parte din anumite clase de echivalentă, sau este similară cu strategia testării restricțiilor și limitelor, unde se testează aplicația folosind valorile limită din datele de input. În ceea ce privește datele de input când vine vorba de o interfață grafică, acestea reprezintă anumite reperuri din interfața grafică, precum introducerea de valori cu anumite proprietăți într-un text field, sau alegerea unui element dintr-o listă de elemente grafice. În urma

acțiunilor de acest gen, testerul își poate da seama în mod manual ce funcționează și ce trebuie îmbunătățit.

În plus față de aceste metode experimentale de testare manuală, mai există tehnici de testare bazate pe stare [MTR08], ce constau în generarea de urme rezultate din interacțiunea manuală cu interfața grafică 2.1, ce sunt folosite ulterior pentru construirea unui automat finit de stări 2.2. Secvențele rezultate de evenimente ce interacționează între ele în cadrul automatului sunt transformate apoi în cazuri de testare corespunzătoare.

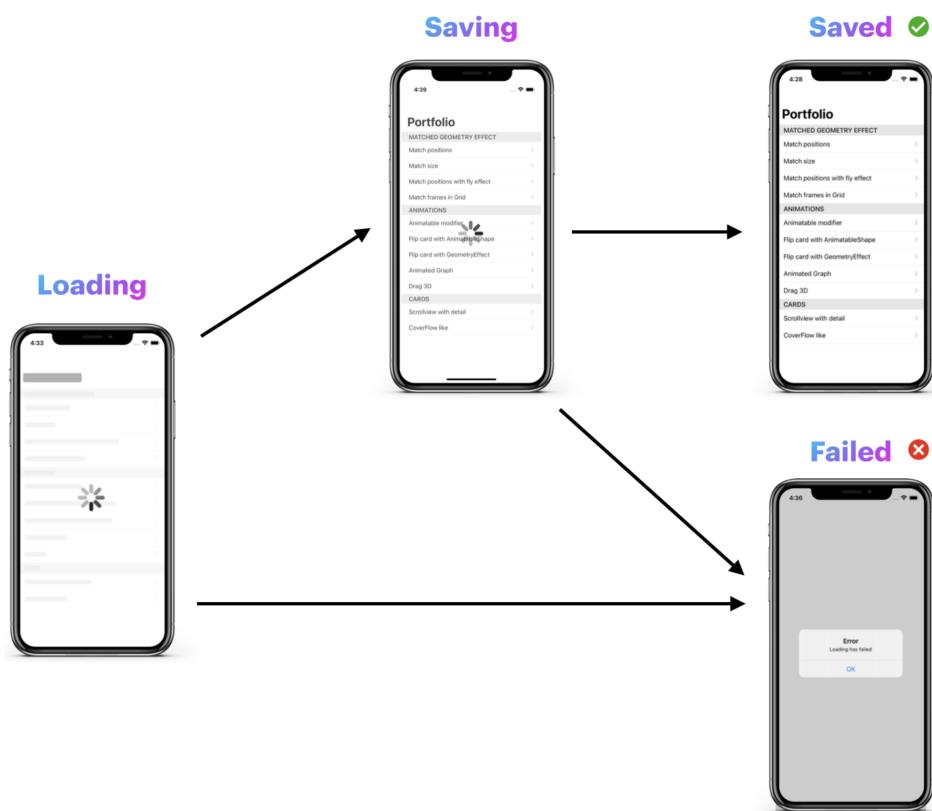


Figura 2.1: Interacțiunea manuală cu interfața grafică [Wit21]

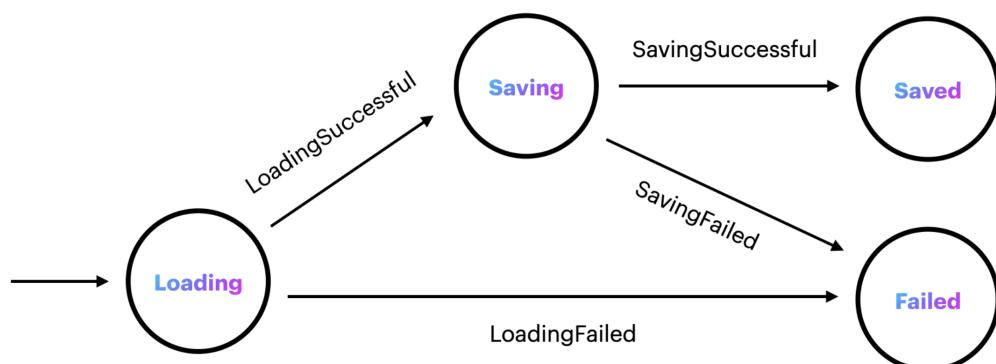


Figura 2.2: Automatul finit de stări generat în urma interacțiunii [Wit21]

În figurile 2.1 și 2.2 se observă trecerea de la un eveniment la altul prin procesul de încărcare a paginii, urmat de cazul valid de salvare a elementului și salvarea ulterioară reușită, respectiv cazul nevalid de eșuare a uneia dintre aceste operații, cazuri de testare ce rezultă din secvențele unui automat finit.

2.2 Automatizarea testării folosind scripturi

Testarea bazată pe scripturi automatizează testarea prin execuția de acțiuni determinate de scripturi asupra elementelor interfeței grafice.

2.2.1 Avantaje și dezavantaje ale testării folosind scripturi

Având în vedere că testarea manuală este prea laborioasă și repetitivă, testerii au venit cu scripturi care automatizează în mod eficient procesul repetitiv de testare prin mimarea acțiunilor utilizatorilor pe interfața grafică. Una dintre dificultățile întâlnite în procesul testării interfețelor grafice este schimbarea acesteia pe parcursul testării, motiv pentru care există diferite abordări ce previn eșuarea testelor din cauza modificărilor apărute pe interfață. De asemenea, există diverse instrumente și limbaje de scripting folosite pe diferite interfețe, precum și framework-uri asociate.

2.2.2 Interfețe desktop, web și mobil

Interfața grafică este în continuă schimbare nu numai datorită apariției de noi funcționalități în aplicație, dar și datorită schimbării interfeței pe care este proiectată inițial aplicația. De exemplu, tranzitia de la aplicație mobilă la pagină web sau tranzitia de la pagină web la aplicație desktop pot cauza schimbări la nivel de poziție de elemente grafice, dar și la nivel de dimensiune a elementelor grafice, ceea ce face ca scripturile inițiale de testare să nu recunoască elementele modificate. Din acest motiv, există metode de reparare a scripturilor de testare ale interfeței grafice ce eficientizează procesul de scriere de scripturi de testare prin păstrarea scripturilor inițiale de testare odată cu schimbarea interfețelor. Acest tip de testare se numește testare de regresie și poate fi efectuată în mod facil folosind abordări precum Reducing Effort in Script-based Testing (REST) [XGF08], Script Repairer (SITAR) [GCZM16], Atomic Maintenance of GUI Test Scripts (ATOM) [LCW⁺17] sau Change-Based Script Maintenance (CHATEM) [CWP⁺18]. Determinarea diferențelor dintre versiunile interfeței grafice pentru schimbarea scripturilor de testare corespunzătoare este un pas important în automatizarea testării folosind scripturi.

2.2.3 Concepte și metodologii de bază

REST [XGF08] este un instrument ce ghidează testerii spre observarea schimbărilor apărute pe interfața grafică și determinarea modului în care pot fi schimbată scriputurile folosite astfel încât testarea să aibă succes. Acesta extrage și compară toate versiunile succesiv lansate ale interfeței grafice și identifică diferențele dintre elementele grafice. Pe baza acestor diferențe, scriputurile de test sunt analizate pentru determinarea referințelor către elementele modificate și REST avertizează în ce locuri trebuie rezolvate erorile din scriputurile de testare.

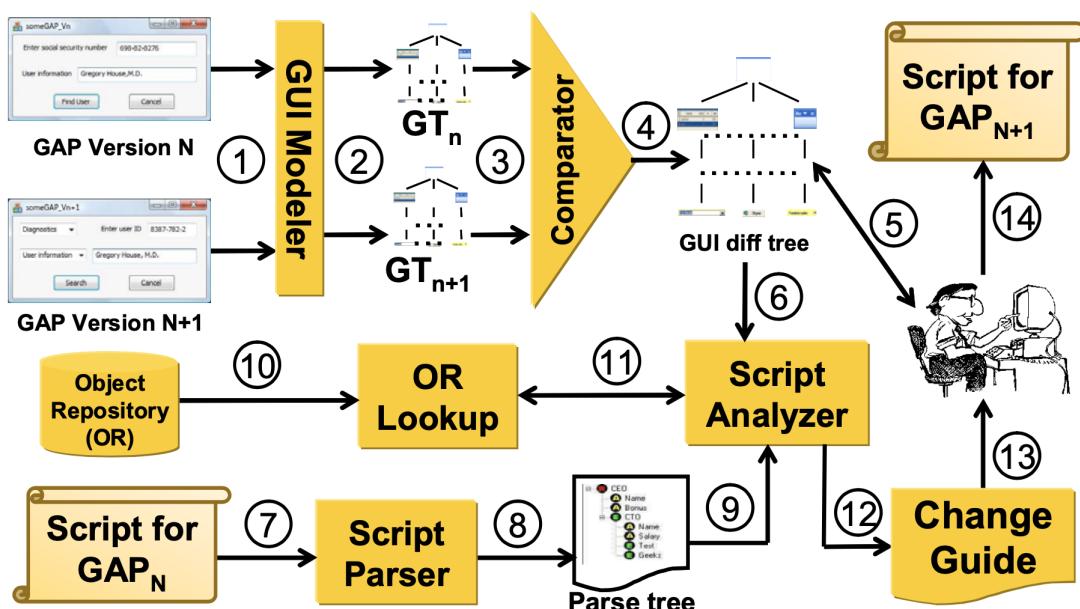


Figura 2.3: Arhitectura REST [XGF08]

Arhitectura REST este afișată în 2.3 și este compusă din secvențe de operații ce primesc drept input două versiuni N și N+1 succesive și diferite ale interfeței grafice. Acestea urmează să fie transformate în arbori care sunt comparați și analizați în scopul furnizării ghidului pentru observarea schimbărilor în interfața grafică și determinarea schimbării scriputurilor corespunzătoare.

SITAR [GCZM16] folosește o abordare diferită față de REST pentru repararea scriputurilor de testare astfel încât acestea să funcționeze și pentru noile versiuni ale interfeței grafice. Metoda propusă diferă prin faptul că nu se cunosc în totalitate interfața grafică și schimbările apărute pe aceasta. Abordarea constă în maparea dintre scriputurile de testare și un event-flow graph adnotat (EFG), ce este un graf constituit din toate interacțiunile posibile dintre evenimentele interfeței grafice.

Arhitectura SITAR 2.4 este formată din Execution space, unde se primesc datele de input, două versiuni N și N+1 succesive și diferite ale interfeței grafice, Script Space, unde sunt date scriputurile de test originale și unde rezultă scriputurile repa-

rate, iar spre Model Space se face maparea dintre scripturi și secvență de evenimente, loc în care se ridică nivelul de abstractizare al celei de-a doua versiuni de interfață grafică pentru a fi permise reparări și se execută repararea scripturilor folosind EFG.

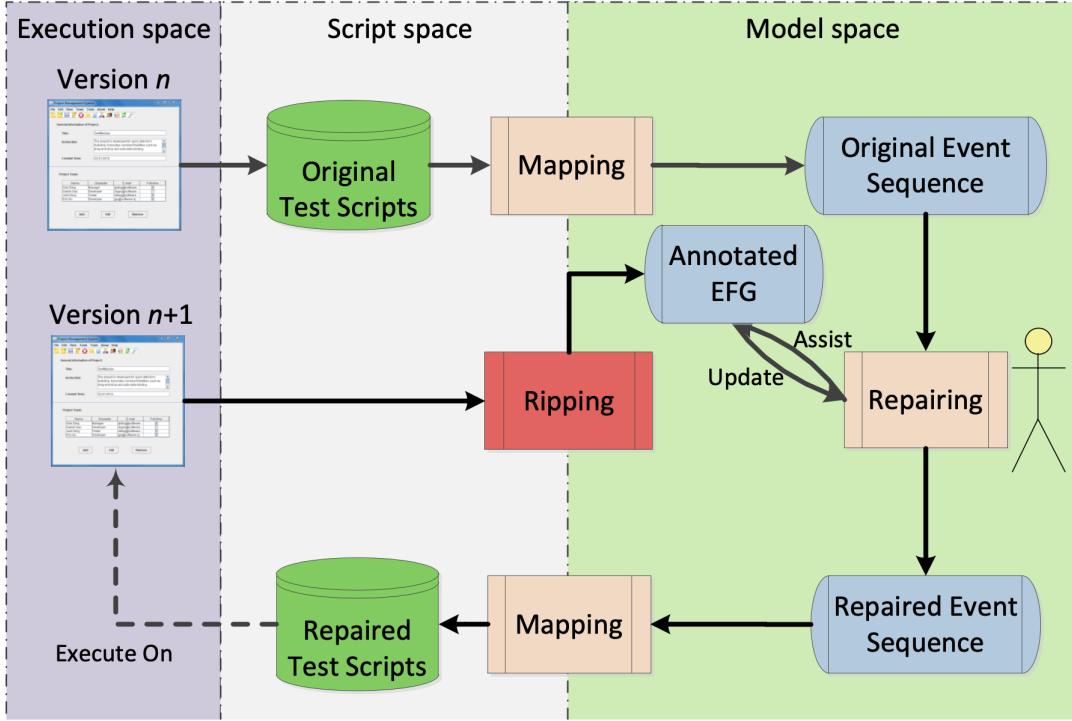


Figura 2.4: Arhitectura SITAR [GCZM16]

ATOM [LCW⁺17] este dezvoltat folosind un event sequence model (ESM), ce este un model format dintr-o secvență de evenimente, și schimbările făcute interfeței grafice sunt salvate într-un delta ESM (DESM). Având ambele modele, ATOM actualizează în mod automat scripturile de test ținând cont de versiunea de bază a interfeței grafice pentru a acomoda schimbările apărute.

Arhitectura ATOM 2.5 evidențiază scripturile de test ale versiunii inițiale prin trecerea lor prin ESM și prin trecerea versiunilor modificate prin DESM, proceduri ce conduc la simulări ale versiunilor și formarea ulterioară a scripturilor reparate în urma modificărilor apărute pe interfață grafică.

CHATEM [CWP⁺18] folosește o abordare asemănătoare cu cea a ATOM, dar mai avansată și mai eficientă deoarece nu mai este nevoie de construirea manuală a DESM precum era nevoie în cazul ATOM. Se dau două ESM pentru versiunea de bază și cea actualizată a aplicației și un grup de scripturi de test pentru versiunea initială. CHATEM extrage automat schimbările dintre cele două versiuni ale interfeței grafice, identifică impactul schimbărilor asupra scripturilor de test, generează și aplică acțiuni de mențenanță asupra scripturilor de test pentru a le actualiza.

Arhitectura CHATEM 2.6 se conturează prin prezența datelor de input și anume

versiunile interfetei grafice, care se transformă în versiuni diferite de ESM și sunt extrase schimbările din acestea, ca în final acele schimbări, împreună cu impactul scripturilor inițiale de test, să se propage spre versiunea finală de scripturi de test.

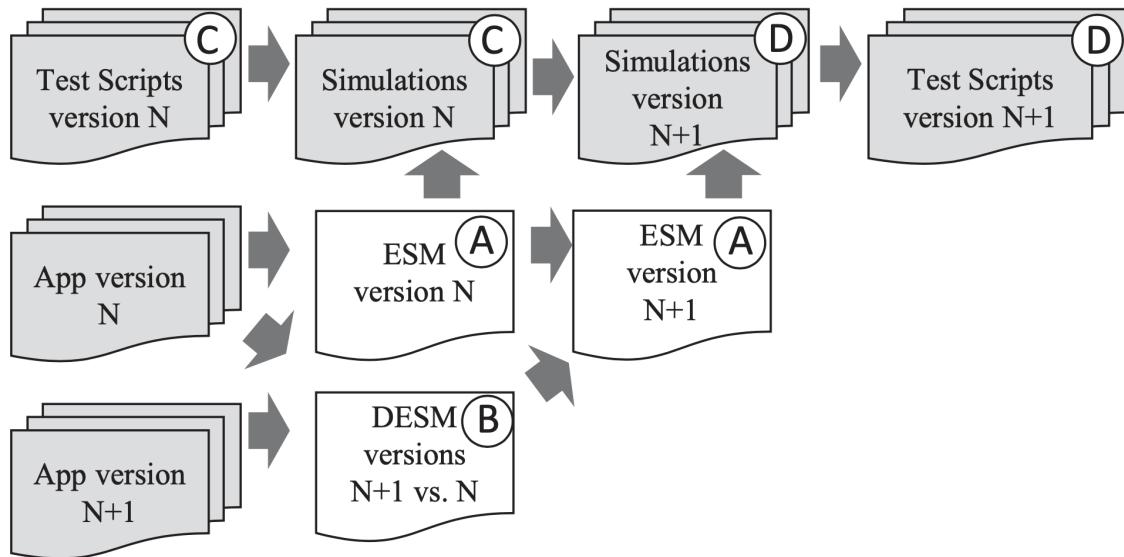


Figura 2.5: Arhitectura ATOM [LCW⁺17]

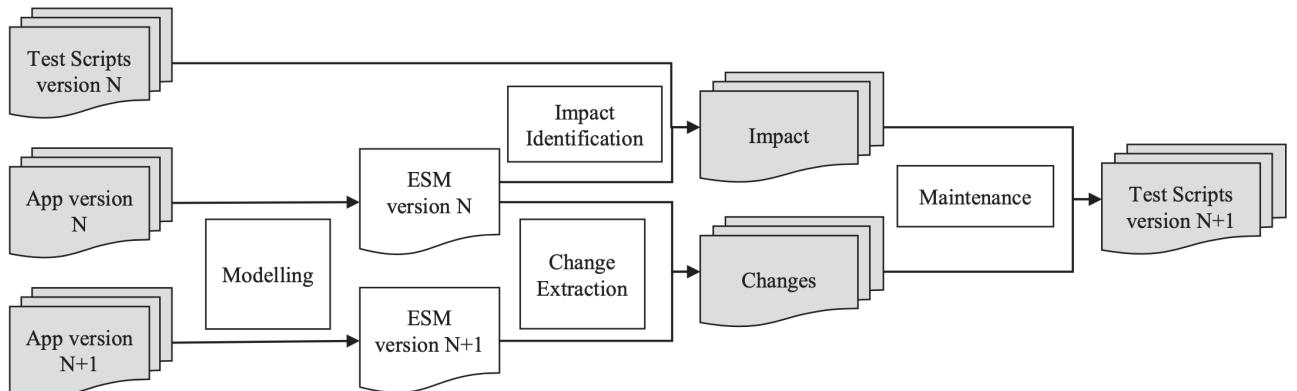


Figura 2.6: Arhitectura CHATEM [CWP⁺18]

2.2.4 Framework-uri existente

În ceea ce privește instrumentele folosite în scrierea testelor bazate pe scripturi, există limbaje de scripting precum Python, JavaScript, Java sau VBScript și framework-uri precum Abbot Costello [Abb11], Selenium [Sel22a], Cypress [Cyp22] și multe altele.

Abbot Costello [SP13] este un framework de testare automată a interfetei grafice prin executarea de scripturi și cazuri de testare programatice în Java. Acesta oferă generare de evenimente și validare pentru componente Swing ale interfetei grafice.

Abbot testează interfața grafică, în timp ce Costello este un editor de scripturi construit peste Abbot ce rulează scripturile de testare în XML. Abbot primește de la Costello fișiere .jar executabile de cod sursă, unde căile către aceste fișiere sunt asociate cu fișiere XML și Abbot generează automat cazurile de testare corespunzătoare. După ce aceste scripturi sunt create, Costello le folosește pentru a rula aplicația, precum este afișat în figura 2.7.

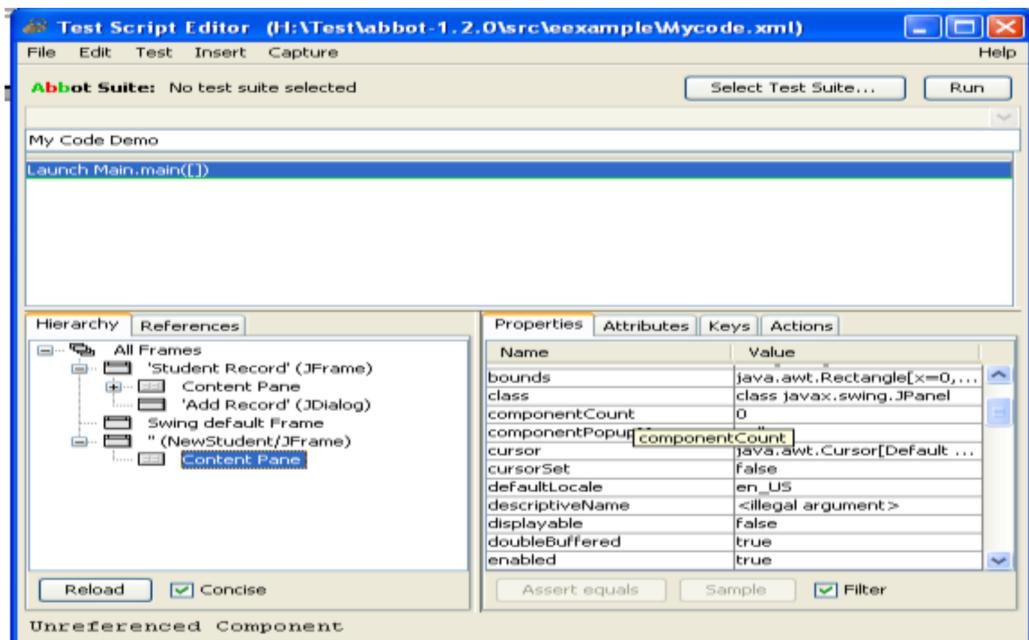


Figura 2.7: Abbot Costello [SP13]

Selenium [RSS17] este compus din mai multe instrumente de testare automată, precum Selenium IDE, Selenium Remote Control (RC), Selenium WebDriver și Selenium Grid. Selenium IDE este un mediu de dezvoltare integrat pentru construirea scripturilor de testare. Acesta este un plugin de Firefox ce permite editarea și depanarea cazurilor de testare. De asemenea, poate înregistra acțiunile utilizatorului și să genereze în funcție de acestea scripturile de testare. Selenium RC a fost primul proiect principal existent, până în momentul în care a apărut Selenium WebDriver (Selenium 2.0), aceasta fiind varianta mai rapidă și mai eficientă ce a rezultat din îmbinarea dintre Selenium RC și WebDriver. WebDriver este o interfață de control de la distanță care permite introspectia și controlul utilizatorilor. Selenium WebDriver comunică în mod direct cu diferite tipuri de browsere, precum și cu aplicații ce folosesc Asynchronous JavaScript and XML (Ajax) - permite paginilor web să fie actualizate asincron prin schimbul de date cu un server web, motiv pentru care este mai rapid decât Selenium RC. Selenium Grid permite executarea scripturilor WebDriver pe dispozitive la distanță (virtuale sau reale) prin direcționarea comenziilor trimise de client către instanțe de browser la distanță. Acesta își propune să ofere o

modalitate usoară de a rula teste în paralel pe mai multe dispozitive. În final, Selenium trece de la varianta 2.0 la varianta 3.0 odată cu adăugarea de noi implementări pentru fiecare tip de browser de la furnizori precum Apple, Google, Microsoft și Mozilla. Deoarece aceștia își cunosc cel mai bine browserele, implementarea lor de WebDriver poate fi cel mai bine cuplată la browser, ceea ce rezultă într-o mai bună experiență de testare. Toată această arhitectură și evoluție a Selenium se regăsește în figura 2.8.

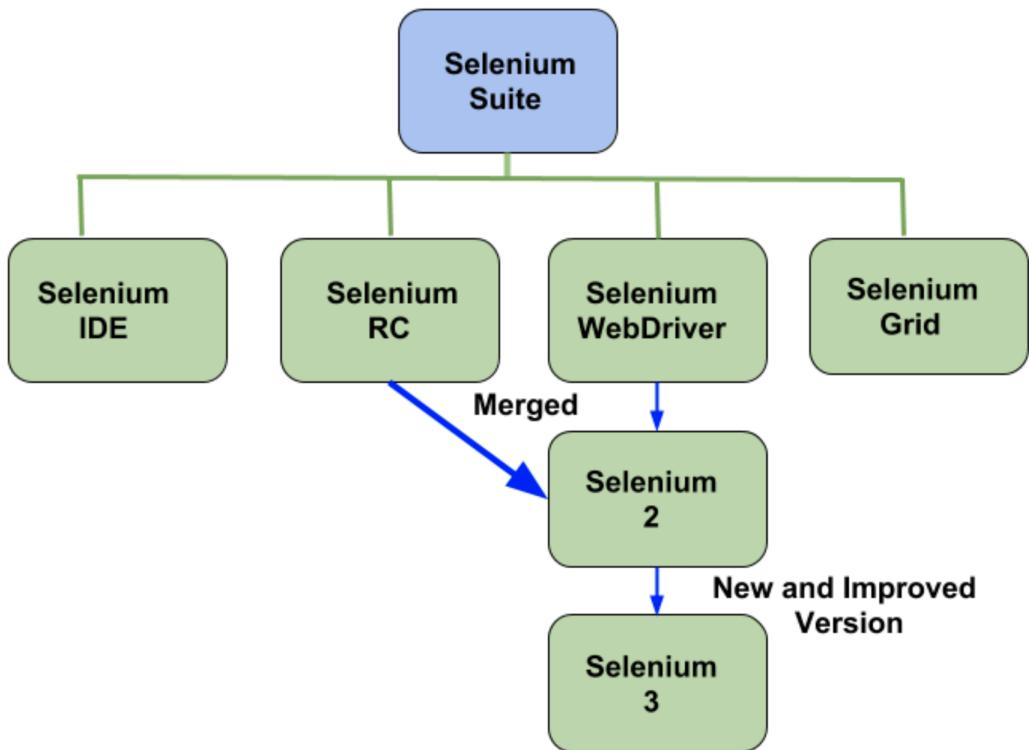


Figura 2.8: Selenium [Jou22]

Majoritatea organizațiilor din industrie folosesc Selenium pentru automatizarea proceselor de testare, fiind un framework foarte cunoscut. Cu toate acestea, Selenium a fost dezvoltat în 2005, când site-urile web erau mult mai simple decât cele din prezent. Principala limitare a Selenium [MA⁺19] este dificultatea de a gestiona elementele web dinamice curente, ceea ce reduce foarte mult performanța testării. Un framework automat de testare ce poate fi folosit pentru aplicațiile prezente este Cypress [Cyp22], a cărui principal avantaj este faptul că simplifică testarea asincronă. Cypress așteaptă încărcarea elementelor de pe interfața grafică și apoi începe să caute elementele grafice corespunzătoare. Aceasta este o limitare pentru Selenium, deoarece testerii trebuie să definească în Selenium manual această așteptare ca pagina să se termine de încărcat. Singurul dezavantaj pe care Cypress îl are este faptul că nu suportă toate tipurile de browser, ci mai mult Chrome, dar având în vedere Google Trends, precum este arătat în figura 2.9, majoritatea oamenilor folosesc

Chrome în majoritate timpului, ceea ce înseamnă că browserul reprezintă o limitare neglijabilă.

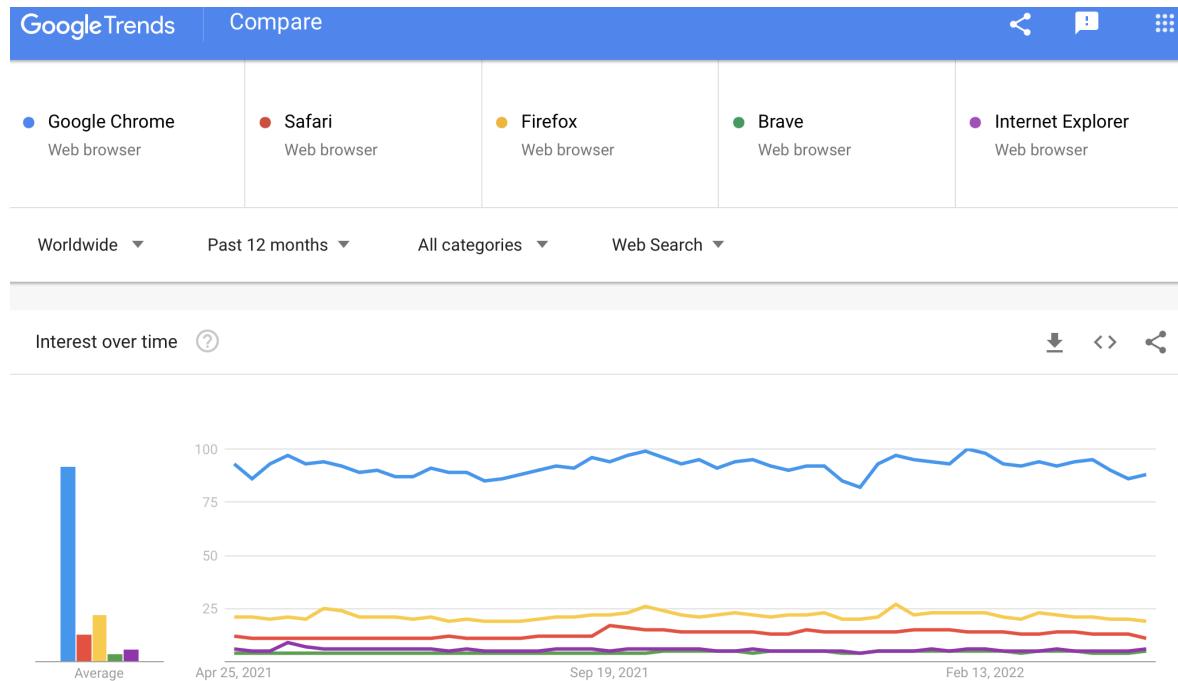


Figura 2.9: Web Browsers Google Trends

Cypress [Cyp22] aduce un plus de valoare testării automate în industrie prin îmbinarea într-un singur framework a tuturor librăriilor și pașilor ce trebuiau făcuți în framework-urile trecute, precum se observă în figura 2.10.

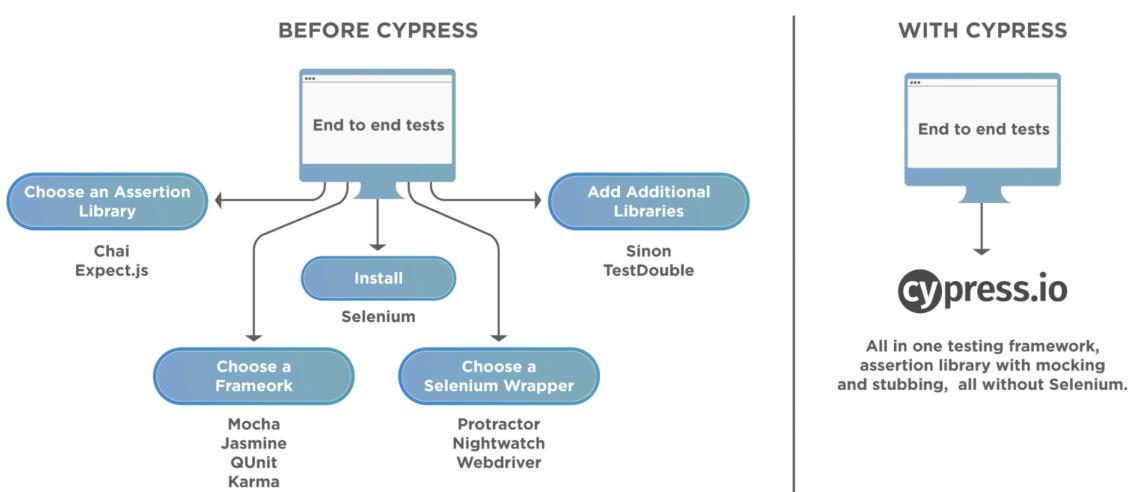


Figura 2.10: Cypress [Khe21]

Majoritatea framework-urilor de testare automată funcționează prin rularea din afară browser-ului și prin executarea de la distanță a comenzi din rețea, dar Cypress [Khe21] operează direct în interiorul browser-ului. Aceasta îi permite framework-

ului Cypress să asculte și să modifice comportamentul browserului în timpul rulării, manipulând DOM și modificând cererile și răspunsurile din rețea în timp real.

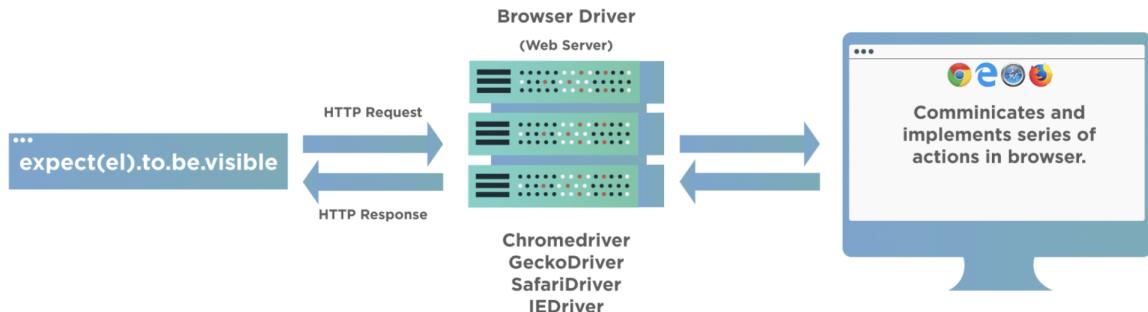


Figura 2.11: Comparație cu Selenium [Khe21]

După cum putem vedea în cazul Selenium în figura 2.11, fiecare dintre browsere și-a furnizat driverele, care interacționează cu instanțele browserului pentru execuțarea comenziilor. Contra acestei metode, în cazul Cypress toate comenzi sunt executate în interiorul browser-ului, precum se poate observa în figura 2.12.

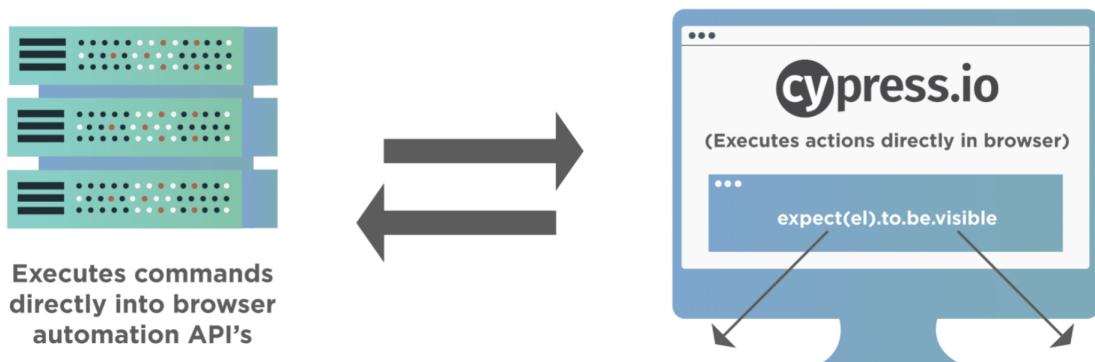


Figura 2.12: Arhitectura Cypress [Khe21]

2.3 Testarea automată

Testarea automată constă în utilizarea de instrumente ce testează în mod automat o aplicație, prin controlarea execuției testelor și compararea rezultatelor reale cu cele prezise, cu scopul de a îmbunătăți calitatea aplicației.

2.3.1 Avantaje și dezavantaje ale testării automate

Testarea automată se concentrează pe domeniul cheie în care suitele de cazuri de testare sunt executate într-un mod mai rapid și repetat în comparație cu rularea manuală a cazurilor de testare, ce necesită mai mult efort din partea testerilor atât

din punct de vedere al timpului, cât și din punct de vedere al nevoii de a observa continuu și atent ce comportament se așteaptă din partea interfeței grafice. Testarea automată desfășoară erorile umane și poate face acțiuni pe care testarea manuală nu le poate face, prin repetiția multiplă și nelimitată a testelor. Pentru ca acest aspect să fie îndeplinit, comparativ cu testarea bazată pe scripturi, testerii nu se mai concentrează pe găsirea de metode de reparare a scripturilor, ci pe găsirea algoritmilor și instrumentelor potrivite, în funcție de interfața pe care aplicația o folosește, pentru automatizarea procesului de testare.

2.3.2 Interfețe desktop, web și mobil

Interfețele desktop [PRZ18] pot fi testate folosind diferite tehnici, precum testarea ce folosește cazuri de testare generate aleator, sau testarea bazată pe modele de interfețe grafică sau bazată pe algoritmi de machine learning (învățarea automată este o ramură a inteligenței artificiale și a informaticii care se concentrează pe utilizarea datelor și a algoritmilor pentru a imita modul în care oamenii învăță, îmbunătățindu-i treptat acuratețea). Cu toate că cele din urmă sunt tehnici mai complexe de testare, un studiu [PRZ18] a arătat faptul că testarea ce folosește cazuri de testare generate aleator acoperă mai bine spațiul de execuție și dezvăluie mai multe erori decât tehniciile mai simple. Acest tip de testare constă în executarea aleatoare a fluxului de evenimente prezente pe interfața grafică, ceea ce generează cazuri de testare ce acoperă diferite secvențe de acțiuni pe interfața grafică. Un instrument automat ce poate fi folosit pentru această tehnică de testare este Testar [VKCF⁺15]. Acesta generează cazurile de testare corespunzătoare prin explorarea iterativă a interfeței grafice, detectând evenimentele și selectând un eveniment aleator ce urmează să fie executat. Testar-Random iterează prin acest proces până în momentul în care atinge o lungime maximă predefinită a cazului de testare și apoi restartează aplicația cu un nou caz de testare. Acțiunile executate de către acest instrument încep de la simple evenimente pe interfața grafică, precum click-uri, până la gesturi mai complexe cu mouse-ul, precum drag-and-drop.

Interfețele web [QWMW17] conțin diferite componente specifice, printre care se numără AJAX, ce îmbină JavaScript cu XML în mod asincron, împreună cu manevrarea de Document Object Model (DOM). Aceasta este o interfață de programare pentru documente web și reprezintă documentul ca noduri și obiecte, precum se poate vedea în figura 2.13, iar în acest fel, limbajele de programare pot interacționa cu pagina web. În plus față de testarea elementelor statice dintr-o pagină web, există și componente dinamice ce trebuie testate. JavaScript este un limbaj orientat eveniment ce permite înregistrarea de event listeners (ascultarea la schimbările produse de către evenimente) la noduri de tip DOM [Mes15].

```

<html>
<head>
<title>
    DOM example
</title>
</head>
<body>
    <p id = "para1">Hello</p>
    <p id = "para2">Hey</p>
</body>
</html>

```

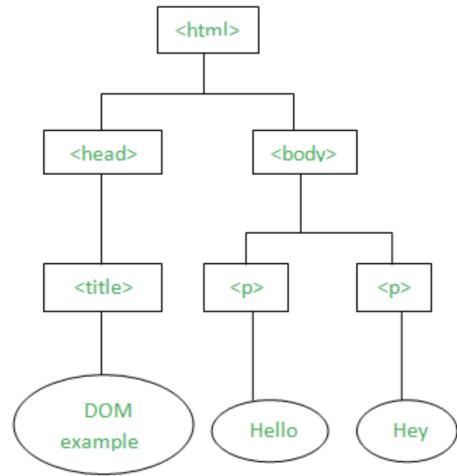


Figura 2.13: DOM [VKCF⁺15]

În timp ce majoritatea evenimentelor pe interfață grafică sunt declanșate de către acțiunile utilizatorilor, apelurile asincrone pot fi declanșate fără input direct de la utilizator. Un singur eveniment poate propaga schimbările către DOM și să declanșeze event listeners mulți, ceea ce face procesul de testare mai dificil. Din acest motiv, aplicațiile web ce folosesc AJAX sunt reprezentate în browser de către un DOM dinamic.

Interfețele mobil [AFT⁺15], precum celelalte tipuri de interfețe, pot fi testate atât cu tehnica de testare aleatorie, cât și folosind testare bazată pe modele sau pe învățare automată. În ceea ce privește testarea aleatorie, este un tip de testare black-box, în sensul în care nu necesită cunoștințe legate de aplicația testată, ci secvențele de evenimente de pe interfață grafică sunt generate aleator. În ceea ce privește tehniciile de testare bazate pe învățarea automată, acestea testează aplicația în timp ce deduc un model al aplicației testate [CNS13].

2.3.3 Concepte și metodologii de bază

O problemă ce poate apărea folosind tehnica deducerii unui model în timpul testării este faptul că adăugarea de noi stări la modelele deja complexe poate duce la o creștere rapidă, posibil chiar exponențială a numărului de stări și tranziții din sistem. Aceasta se numește „problema exploziei de stare” [CKNZ11] și este o problemă foarte cunoscută în domeniu. Pentru a face față acesteia, se folosesc strategii pentru a lega setul de intrare, care în schimb poate limita acoperirea și capacitatea de detectare a defectiunilor tehnicii de testare. Printre aceste strategii se numără procesul de inginerie inversă [MP18], care este responsabil pentru analizarea stării aplicației și interacțiunea cu aceasta prin declanșarea evenimentelor. În acest sens,

se produce modelarea stării interfeței grafice prin definirea unui criteriu adecvat de testare bazat pe un automat de stări. Aceasta este o tehnică de generare de cazuri de testare ce folosește anumite modele și criterii. O altă strategie este potrivirea modelelor, ce încearcă să identifice prezența modelelor de interfață grafică pe baza unui catalog de modele. Când este detectat un model din catalog, se aplică o strategie de testare. Aceste strategii de testare se numesc UI Test Patterns, unde modelele sunt identificate și testate între declansarea evenimentelor, adică procesul alternează între explorarea aplicației și testarea modelelor de interfață grafică. Procesul este dinamic și complet automat, nu necesită cunoștințe anterioare despre aplicația testată.

2.3.4 Framework-uri existente

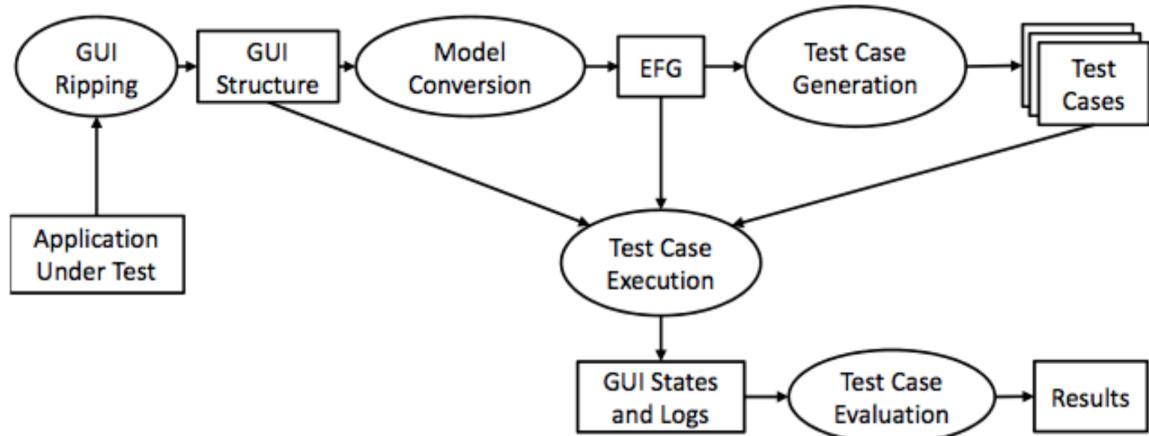


Figura 2.14: Arhitectura Guitar [Sou14]

În ceea ce privește interfețele desktop, tehniciile de testare bazate pe modele au la bază grafuri construite dinamic în timpul explorării spațiului de testare. Nodurile grafurilor corespund cu stările interfeței grafice, iar muchiile reprezintă tranzitiiile dintre stări. Instrumentele specifice acestei tehnici de testare sunt GUITAR-EFG, o tehnică implementată în cadrul framework-ului Guitar [NRBM14]. Acesta folosește un event flow graph (EFG) ce este construit automat prin explorarea interfeței grafice, sau GUITAR-EIG, o versiune ce se bazează pe un event interaction graph (EIG) unde sunt generate cazuri de testare mai relevante, sub forma de secvențe de evenimente filtrate cu evenimente precum închiderea și deschiderea de ferestre. Mai există GUITAR-EDG, ce este o versiune Guitar care depinde de un event dependency graph (EDG), în care sunt generate cazuri de testare formate din îmbinarea de evenimente ce sunt inter-dependente între ele. Arhitectura Guitar poate fi observată în figura 2.14.

Interfețele web diferă prin structură, drept urmare au nevoie de alte instrumente de testare. Studii anterioare [OJPZ11] conduse pe baza erorilor întâlnite în top 100

cele mai vizitate site-uri web, au demonstrat faptul că majoritatea erorilor de JavaScript întâlnite au legătură cu probleme în DOM ce sunt cauzate de actualizări greșite ale acestuia. Drept urmare, autorii sugerează ca eforturile de testare să se concentreze pe detectarea de probleme legate de DOM. O abordare de testare a interfețelor grafice ce au la bază AJAX [MVDL12] constă în detectarea și executarea automată de event listeners ale interfeței grafice ce conduc la diferite stări ale DOM-ului dinamic corespunzător aplicației web. Acest obiectiv poate fi atins prin folosirea instrumentului Crawljax, care generează un State Flow Graph ce capturează stările interfeței grafice și posibilele tranziții bazate pe evenimente ce se petrec între stări, în urma analizării DOM-ului înainte și după declanșarea unui eveniment, precum este arătat în figura 2.15. Acest State Flow Graph poate fi supus generării de teste automate corespunzătoare.

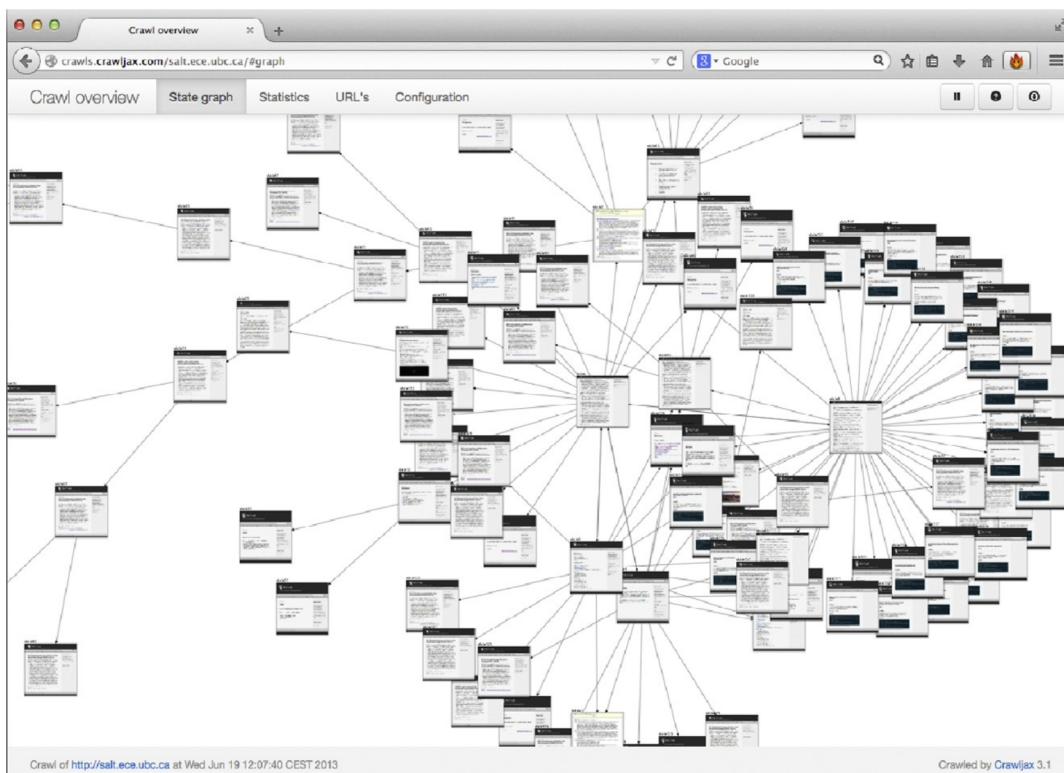


Figura 2.15: Crawljax [Mes15]

Când vine vorba de interfețe mobil, instrumentele ce pot fi folosite pentru testarea aleatorie sunt Monkey [Stu22] sau Dynodroid [MTN13] pentru Android și SwiftMonkey [Man17] pentru IOS. Pe de altă parte, testarea bazată pe modele necesită un model al aplicației pentru generarea de input corespunzător, iar o suiată de instrumente ce poate fi folosită în acest sens este TEMA [TKH11] pentru Android și Dribble [Dri22] pentru IOS. Testarea pe bază de învățare automată se poate face folosind MobiGUITAR [AFT⁺15], un instrument bazat pe AndroidRipper [AFT⁺12], o versiune care în mod automat face inginerie inversă a automatului de stări din

aplicație. Mai exact, cazurile de testare sunt generate pe baza analizei dinamice a interfeței grafice și găsirea evenimentelor corespunzătoare acesteia.

2.4 Testarea folosind Record and Playback

Testarea folosind Record and Playback este un tip de testare automată în care se înregistrează activitatea utilizatorului și apoi este imitată. Acesta [Tes19] permite crearea, editarea și importarea scripturilor înregistrate și, de asemenea, urmărirea proceselor de testare.

2.4.1 Avantaje și dezavantaje ale testării Record and Playback

Testarea record and playback [Mes03] poate fi mai eficientă ca testarea bazată pe scripturi, deoarece modificarea interfeței grafice necesită refactorizarea codului folosit în testarea automată, în timp ce testarea record and playback înregistrează acțiunile efectuate asupra interfeței grafice și nu trebuie făcute modificări manuale asupra codului. Este facilă modificarea sau schimbarea scripturilor înregistrate oriunde sunt găsite erori sau greșeli în mijlocul procesului de înregistrare. Acest lucru face ca procesul să fie mai rapid și mai flexibil, fără a fi nevoie de reînregistrarea întregului script, precum se poate observa în figura 2.16.

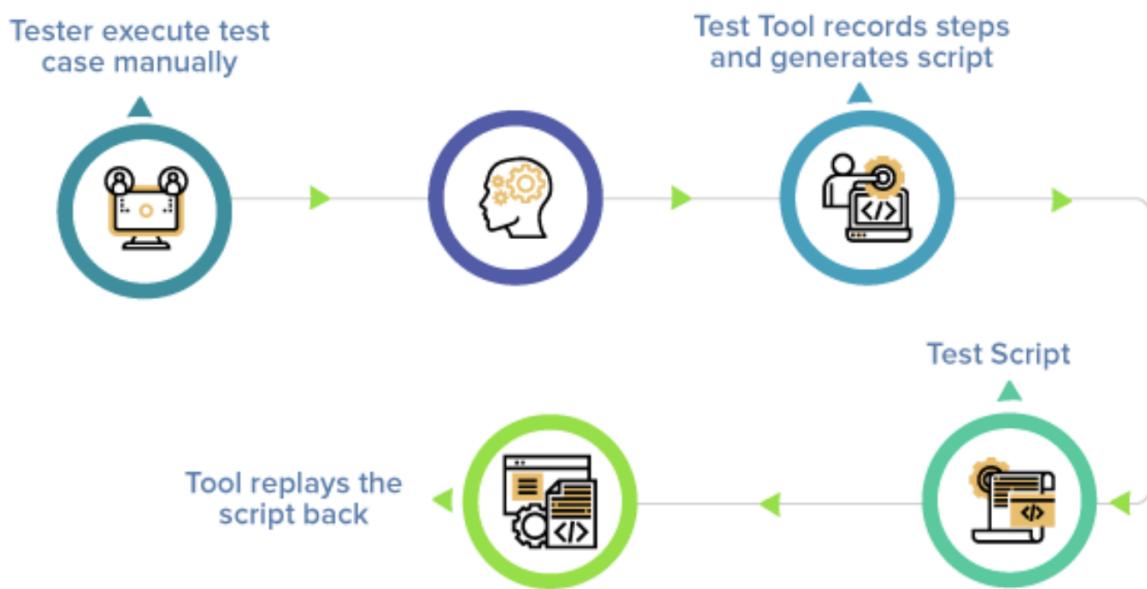


Figura 2.16: Testarea record and playback [Tes19]

2.4.2 Concepte și metodologii de bază

Testarea record and playback este formată din partea de recording și anume înregistrarea acțiunilor utilizatorului pe interfața grafică și din partea de playback ce constă în derularea repetată ale acțiunilor înregistrate. Pentru a exemplifica metodologiile de bază legate de testarea record and playback, HttpUnit [Mes03] poate fi folosit pentru emularea acțiunilor în browser.

Test Recording:

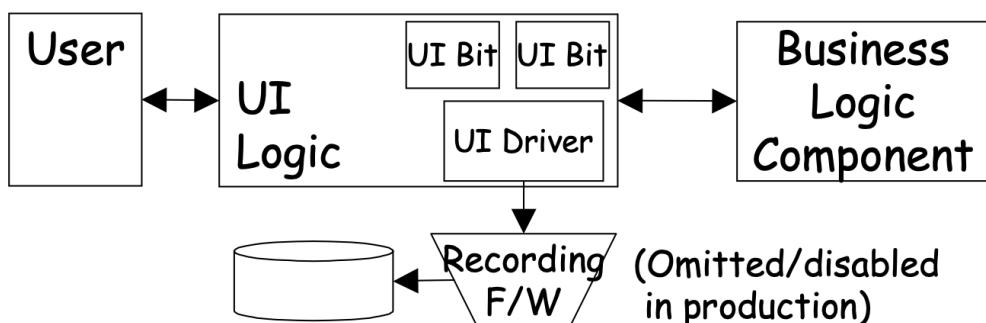


Figura 2.17: Test Recording [Mes03]

Test Execution:

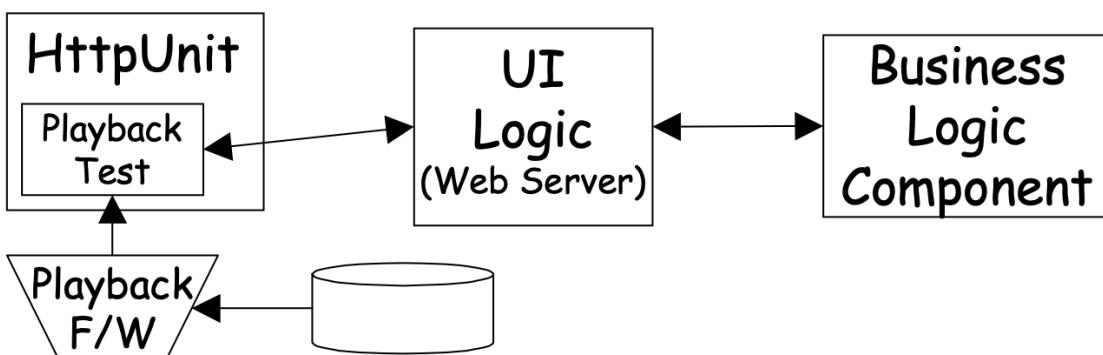


Figura 2.18: Test Execution [Mes03]

În ceea ce privește partea de record, se înregistrează acțiunile utilizatorului asupra interfeței grafice și se salvează într-un fișier. Pentru partea de playback, HttpUnit citește un element *< interaction >* din fișierul de record și trimite conținutul elementului *< request >* către serverul web. Apoi ia HTML-ul pe care l-a primit înapoi și îl compară cu, conținutul elementului *< expected - html >* din sesiunea înregistrată. Dacă se potrivesc, citește următorul element *< interaction >* din fișier. Când ajunge la sfârșitul fișierului fără ca nicio comparație să esueze, testul trece cu succes. Acum procesul de recording, urmat de playback și execuția testelor se

poate observa în figurile 2.17 și 2.18. Atunci când sunt produse schimbări la nivelul interfeței grafice, mai exact schimbarea HTML-ului, trebuie rulate manual testele cu partea de recording pornită. Odată ce s-a verificat că rezultatele sunt corecte, se înlocuiește fișierul original de playback (ce include rezultatele așteptate) cu noul fișier de record.

2.4.3 Framework-uri existente

Există diferite framework-uri și instrumente ce pot fi folosite în testarea record and playback. În ceea ce privește framework-uri mai avansate de testare ce folosesc metoda de record and playback, cele mai întâlnite sunt Quick Test Professional și Test Complete. De asemenea, și Selenium are integrată această componentă de testare, cu toate că este mai mult folosit pentru testarea automată. Selenium IDE are o funcție de înregistrare, care înregistrează acțiunile utilizatorului pe măsură ce sunt efectuate și apoi le exportă ca un script reutilizabil, care poate fi executat ulterior, într-unul dintre multele limbaje de programare. Redarea repetată se face astfel încât procesul de testare să fie efectuat de câte ori este necesar. Cu toate acestea, această tehnică de înregistrare și redare ar trebui evitată atunci când este de așteptat ca comportamentul sistemului să se schimbe semnificativ între momentul în care testele sunt înregistrate și când vor fi redate [ND12].

Quick Test Professional (QTP) înregistrează toate acțiunile efectuate de utilizator, dar nu oferă acces ușor la comenzi. Când se apasă butonul de înregistrare, aplicația este pornită. Dar nu se poate introduce punctul de control în timpul înregistrării. Acestea se pot introduce numai după înregistrare. QTP oferă trei tipuri de înregistrare, care sunt modul sensibil la context, modul analog și înregistrarea la nivel scăzut. Nu există nicio modalitate de a întrerupe testul la mijloc. Deci, 99% din testeri folosesc modul sensibil de context, deoarece stochează doar acțiunile aplicației, ignorând mesajele de eroare ale sistemului. QTP și TC generează amândouă documentația automată a acțiunilor efectuate de utilizatori. Dar în comparație cu TC, QTP generează doar VbScripts. După execuția scriptului de testare, este necesar să se obțină rezultatele execuției pentru efectuarea unei analize eficiente, dacă scripturile de testare au eșuat în timpul rulării unei suite de teste. QTP oferă un rezumat executiv al testului. Afisează etapele de testare în arborele ierarhic și oferă, de asemenea, rezumatul fiecărui pas de testare. De asemenea, oferă informații despre punctele de control care au fost aplicate în timpul testării. QTP oferă statisticile despre rularea precedentă și rularea curentă sub formă de diagrame pie [KK11]. Aceste rapoarte de rezultate sunt foarte ușor de utilizat și ușor de înțeles, precum se poate observa în figura 2.19.

Având în vedere testarea folosind TestComplete (TC) [Sma21a], acesta este un

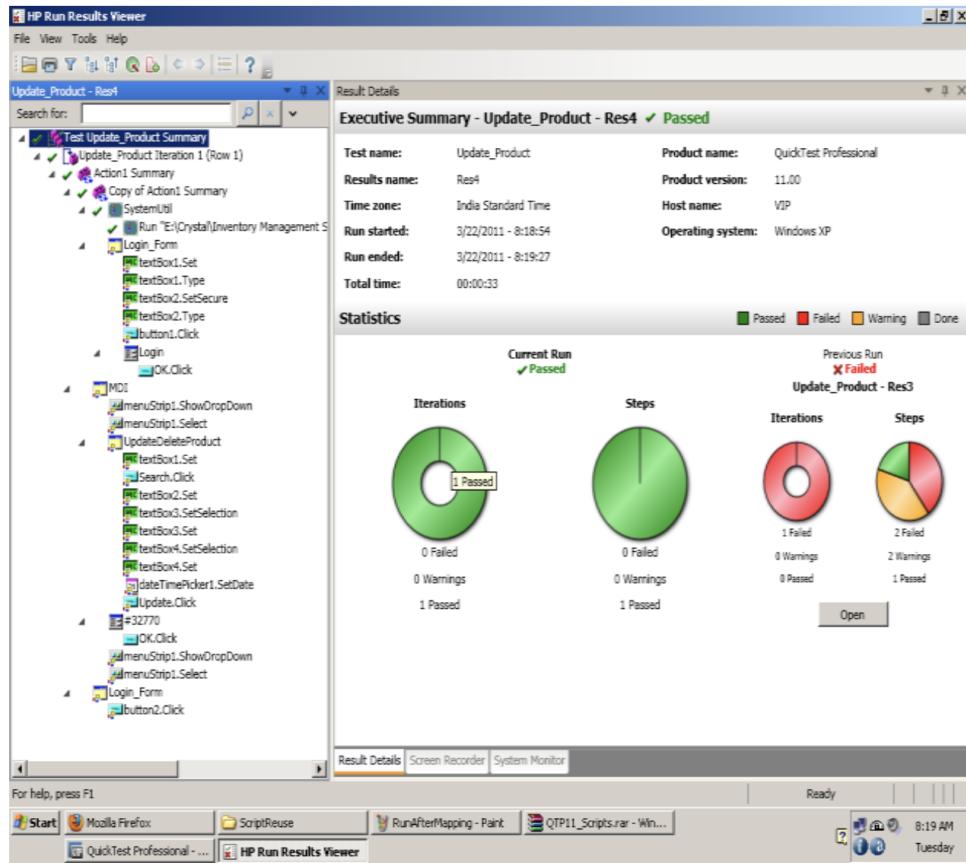


Figura 2.19: Rezultate execuție QTP [KK11]

framework ce oferă acces foarte ușor la comenzi 2.20. În timpul înregistrării, motorul de înregistrare TC este întotdeauna prezent în aplicație și clipește, ceea ce înseamnă că înregistrează acțiunile utilizatorului. Bara de instrumente de înregistrare are toate comenzile. Deci, se pot aplica cu ușurință punctele de control, se poate adăuga text și, de asemenea, se pot vedea coordonatele ecranului și coordonatele ferestrei. TC oferă diferite tipuri de înregistrare, cum ar fi cuvinte cheie, script, procedură de nivel scăzut bazată pe coordonatele ecranului sau ferestrei și pe HTTP. TC oferă o modalitate usoară de a întrerupe înregistrarea la mijloc. Deci, se poate gestiona aplicația testată sau schimba mediul fără a înregistra acele acțiuni în script. De asemenea, TC poate genera cinci tipuri de scripturi, precum VbScript, Delphi, C++, C# și JScripts. Dacă o aplicație se bazează pe oricare dintre ele, atunci TC poate genera cu ușurință scripturile corespunzătoare [KK11].

TC oferă, de asemenea, informații despre fiecare pas de testare și o reprezentare grafică a rezultatelor, precum se poate vedea în 2.21.

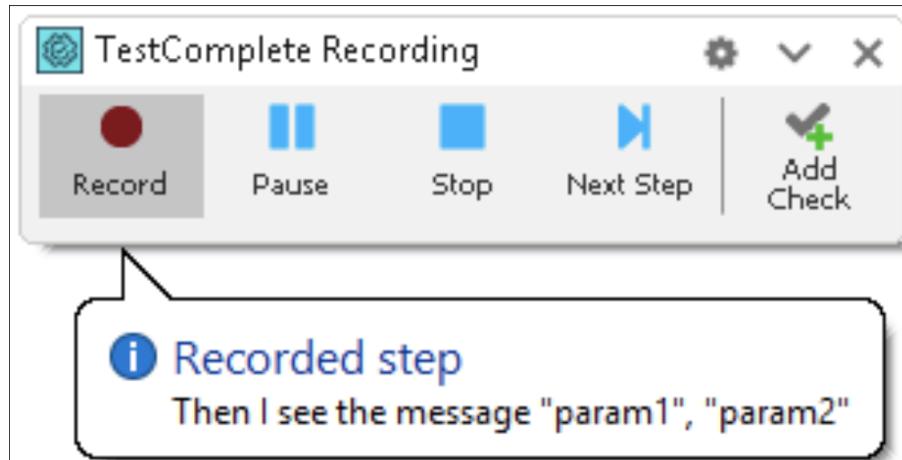


Figura 2.20: TestComplete Recording [Sma21a]

2.5 Testarea folosind Robotic Process Automation

Robotic Process Automation (RPA) este tehnologia care oferă instrumente pentru automatizarea acțiunilor și proceselor executabile la nivelul proceselor de business, dar și la nivelul de interfață grafică. Automatizarea este efectuată de către robotii software programati să execute acțiunile repetitive ale oamenilor, precum testarea interfeței grafice de exemplu. Aceștia pot executa click-uri pe diferite elemente ale interfeței grafice, completa field-uri cu text, extrage date structurate sau nestructurate din diferite aplicații prin examinarea fișierelor sau a interfețelor grafice și pot folosi chiar și Application Programming Interface (API - intermediar software care permite ca două aplicații să comunice între ele) dacă este necesar.

2.5.1 Avantaje și dezavantaje ale testării Robotic Process Automation

Testarea folosind RPA este mai eficientă din punct de vedere al timpului comparativ cu testarea automată, deoarece odată ce robotii software sunt programati, aceștia parcurg toate cazurile repetitive de testare fără a avea nevoie de asistență. Unul dintre cele mai mari avantaje este că robotii pot lucra 24/7 non-stop, scăzând astfel timpul consumat de resursele umane la aceleași procese. Atunci când sunt adaptați în mod eficient la procesele care sunt automatizate, robotii pot procesa cantități mai mari de muncă cu un risc mai mic de eroare și ale căror rezultate pot fi, de asemenea, înregistrate în același mod în care le-ar face un testator uman, ceea ce se traduce printr-o mai bună calitate a informațiilor care sunt prezentate testerilor umani. De asemenea, sunt mai puțini pași ce trebuie parcursi folosind RPA, precum se poate observa în figura 2.22. În timp ce testarea automată necesită decizia de a folosi testare automată, achiziționarea instrumentelor potrivite în funcție de aplicație,

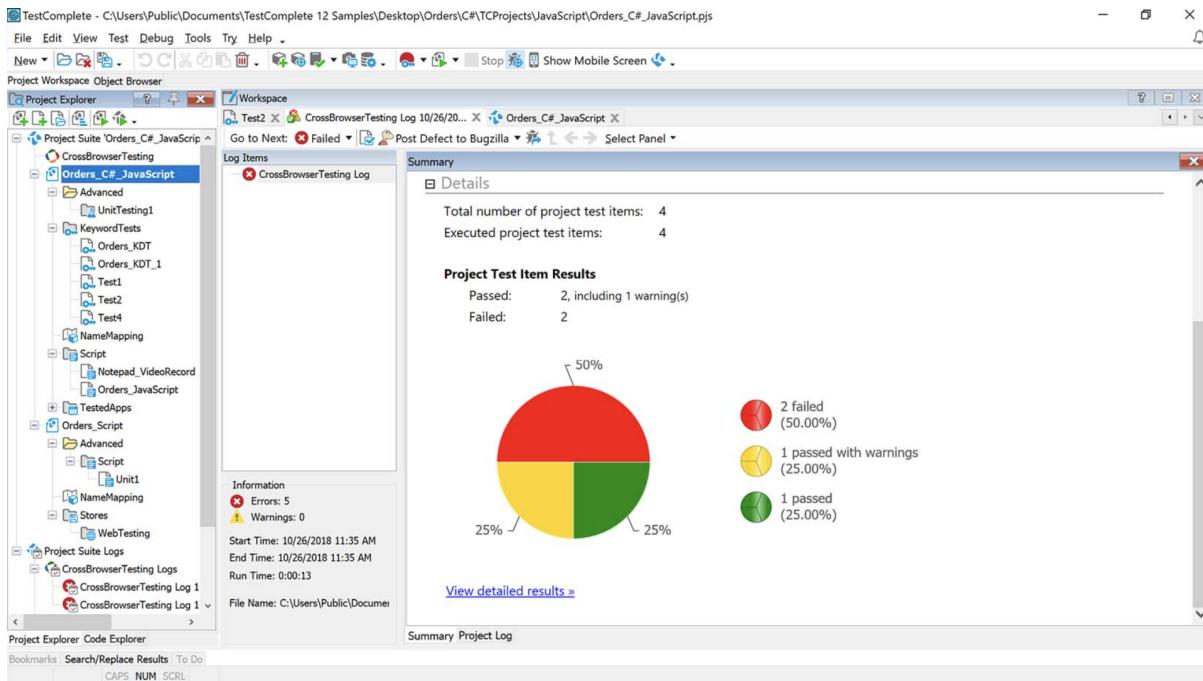


Figura 2.21: Rezultate execuție TC [Sma21b]

introducerea în procesul de automatizare, partea de review de cod, execuția și administrarea testelor, planificarea testelor și deciziile de dezvoltare, RPA necesită doar analizarea aplicației, dezvoltarea mediului necesar testării, testarea și partea de deployment și mențenanță.

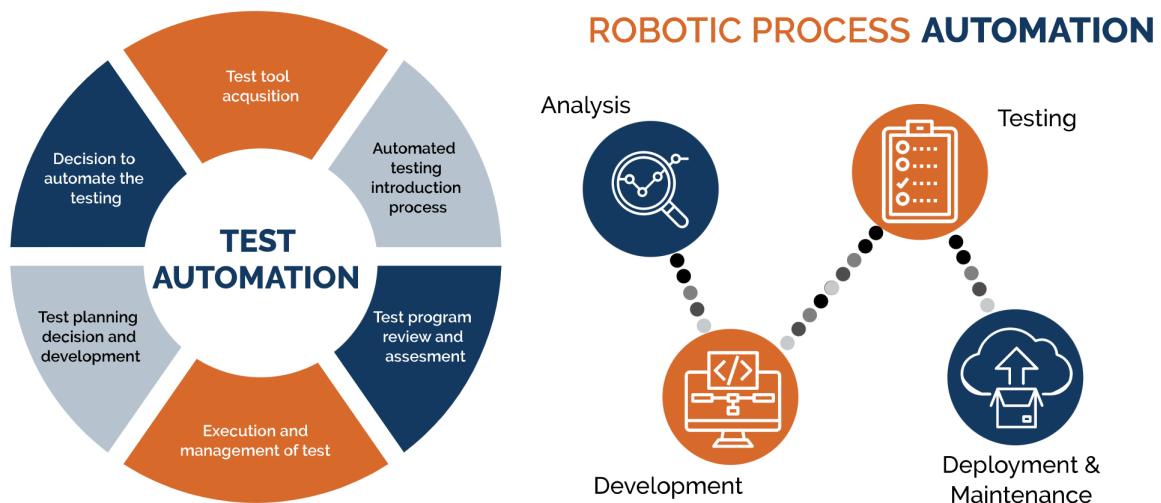


Figura 2.22: Comparativ cu testarea automată [Roy20]

2.5.2 Framework-uri existente

Furnizorii RPA [CSS20] oferă platforme pentru definirea și operarea unor astfel de roboti software. Platformele RPA conțin cel puțin următoarele trei componente:

o componentă pentru a descrie sau modela robotii (acest lucru se face de obicei vizual), o componentă pentru a executa și integra robotii în mediu și în aplicațiile existente și în final, o componentă, numită de obicei orchestrator, care implementează robotii, programează și monitorizează execuția acestora. În plus, soluțiile avansate RPA oferă și capabilități de inteligență artificială (AI), cum ar fi procesarea limbajului natural, învățarea automată și viziunea computerizată, pentru a procesa mai bine intrările textuale sau vizuale ale robotului. De asemenea, robotii pot fi supravegheati (cei care necesită intervenție umană din când în când) sau nesupravegheati (funcționând independent). Aceștia pot rula local, în cloud sau într-un mediu virtual.

În ultimii ani, următorii trei furnizori de instrumente RPA - UiPath [UiP22c], Automation Anywhere [Aut22] și Blue Prism [Blu22] au fost desemnați lideri în domeniu [LCOLL19]. Platforma UiPath este principalul furnizor de instrumente RPA, bazat pe dimensiunea și evaluarea companiei, pe cota de piață, precum și pe amploarea și profunzimea soluției, iar în plus cercetările anterioare [CSS20] au demonstrat că are cel mai avansat suport de testare RPA, motiv pentru care acesta este cea mai relevantă platformă RPA pentru explorarea soluțiilor de testare oferite.

2.5.3 Concepțe și metodologii de bază

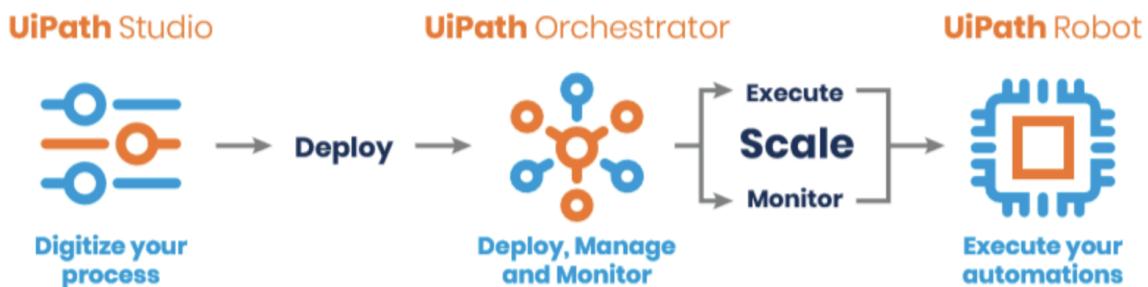


Figura 2.23: Platforma UiPath [UiP22b]

Componenta în care sunt proiectați robotii se numește UiPathStudio [UiP22d]. Ordinea în care activitățile pot fi efectuate de un robot poate fi modelată în UiPath Studio cu trei tipuri principale de diagrame de flux de lucru [UiP22e]: secvență, diagramă de flux, mașină de stări și un instrument de gestionare a exceptiilor. Secvența este o reprezentare liniară simplă a activităților care urmează una după alta. Diagrama de flux adaugă mai multă flexibilitate prin decizii și săgeți între orice activități. Este similar cu diagramele de activitate UML. În cele din urmă, mașina de stări este chiar mai expresivă decât o diagramă, permitând tranziții condiționate. Ele sunt similare cu mașinile de stare clasice UML. Construcțiile de mai sus pot conține și date sub formă de argumente de diagramă și variabile locale. De asemenea, toate diagramele pot fi încorporate ierarhic unele în altele, de exemplu, poate

există o secvență de organigrame care conțin mașini de stare locale în anumite noduri. Apoi, UiPathStudio comunică cu Orchestrator pentru partea de deployment, de unde sunt programati și monitorizați roboții software, precum se poate vedea în figura 2.23.

UiPath a lansat o soluție numită UiPath Test Suite [UiP22b], care este printre primele de acest fel în domeniul RPA. Această soluție oferă managementul testelor prin organizarea suitelor de testare, execuția testelor și raportarea testelor. Acceptă testarea RPA, testarea mobilă, a aplicațiilor și testarea API prin proiectarea cazului de testare bazat pe date. Pentru testarea RPA, UiPath a creat un şablon de caz de testare ca flux de lucru al secvenței dedicate care conține alte trei subsecvențe numite Given, When și Then [UiP20], corespunzătoare pregătirii testului, execuției testului și evaluării testului, precum este afișat în figura 2.24

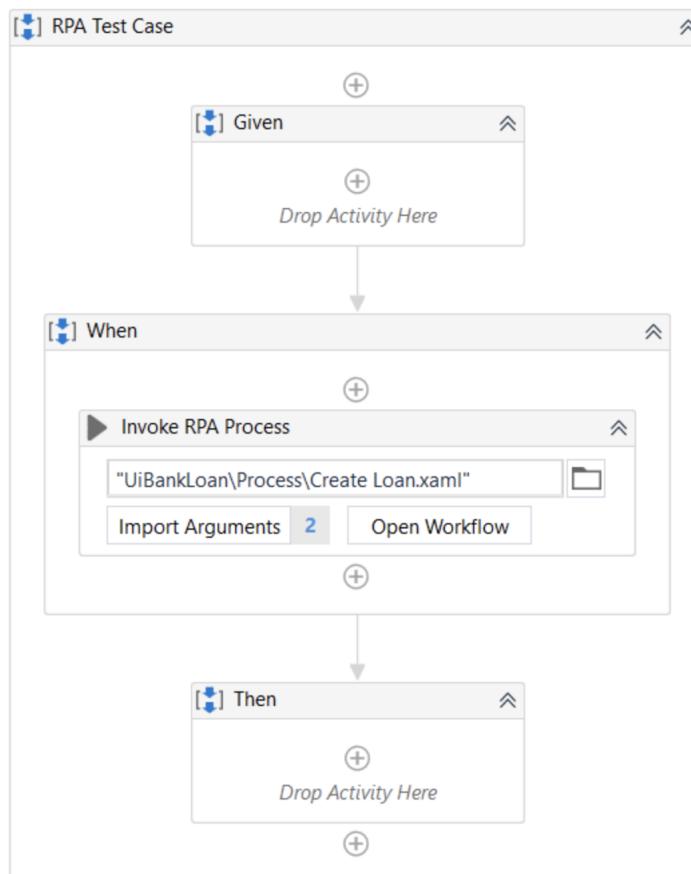


Figura 2.24: Caz de testare UiPath [UiP20]

La testarea produsă de un robot, va fi creată o suită de teste cu mai multe cazuri de testare, acoperind diverse scenarii. O caracteristică a UiPath Test Suite este că, după executarea suitei de testare, testerul primește acoperirea testului ca procent din activitățile acoperite ale robotului, evidențiind-le vizual 2.25.

Comparativ cu o metodă tradițională de Record and playback folosită de obicei în industrie pentru a testa interfețele grafice, instrumentul de la UiPath oferă

CAPITOLUL 2. TESTAREA INTERFETELOR GRAFICE

TEST RUN	VARIATION	RESULT	VERSION	ROBOT	MACHINE	STARTED	ENDED
Create Account		Passed	6.7.8	Emerald Koe	DESKTOP-4Q3HFP2	a minute ago	a minute ago
Create Loan		Failed	6.7.8	Emerald Koe	DESKTOP-4Q3HFP2	a minute ago	a minute ago
Create User		Passed	6.7.8	Emerald Koe	DESKTOP-4Q3HFP2	a minute ago	a minute ago
Delete Account		Passed	6.7.8	Emerald Koe	DESKTOP-4Q3HFP2	a minute ago	a minute ago
Modify User Profile		Passed	6.7.8	Emerald Koe	DESKTOP-4Q3HFP2	a minute ago	a few seconds ago
Perform Transaction		Passed	6.7.8	Emerald Koe	DESKTOP-4Q3HFP2	a few seconds ago	a few seconds ago
Register Administrator		Failed	6.7.8	Emerald Koe	DESKTOP-4Q3HFP2	a few seconds ago	a few seconds ago
Reset Password		Passed	6.7.8	Emerald Koe	DESKTOP-4Q3HFP2	a few seconds ago	a few seconds ago
Transfer Funds		Passed	6.7.8	Emerald Koe	DESKTOP-4Q3HFP2	a few seconds ago	a few seconds ago

Figura 2.25: Suta UiPath de teste [UiP22b]

nu numai partea de programare a robotilor pentru repetitia actiunilor, dar si partea de recunoastere a elementelor de pe interfetele grafice folosind computer vision, impreună cu metoda de Record and playback inclusă în platformă, fiind o metodă mult mai completă și mai intuitivă de testare a interfetelor grafice. De asemenea, metricile sunt mult mai sugestive, statistica RPA arătând câte procese au fost automatizate, rata de succes, productivitatea robotilor și grafice pentru timpul mediu folosit în testare, metriki mult mai complete din punct de vedere al performanței, datorită instrumentului Insights [UiP22a], ilustrat în figura 2.26.

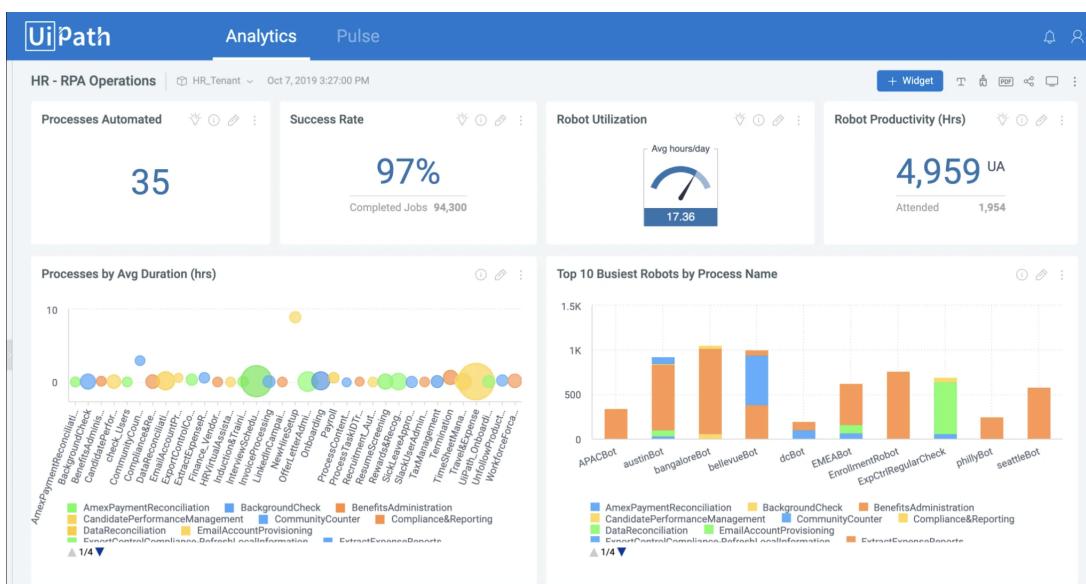


Figura 2.26: Grafice de performanță UiPath [UiP22a]

2.6 Dificultăți în testarea interfețelor grafice

În urma explorării tuturor acestor tehnici de testare ale interfeței grafice, diferite dificultăți pot fi întâlnite în funcție de metoda folosită, împreună cu instrumentul folosit și tipul de interfață ce se testează.

Pentru început, testarea manuală poate fi dificilă pentru un tester neexperientat având în vedere că de multe ori tehniciile de testare folosite în cadrul acestei metode sunt mai mult bazate pe experiența anterioară. În plus, chiar și testerii experimentați pot comite greșeli folosind această tehnică, deoarece erorile umane nu pot fi evitate, iar soluția pentru această problemă ar fi automatizarea procesului de testare. Primele încercări de automatizare a testării au venit odată cu folosirea de scripturi de testare. Noua problemă apărută în folosirea scripturilor de testare a fost neconcordanța scripturilor cu modificările apărute pe interfața grafică, fie din cauza adăugării de noi funcționalități, fie din cauza schimbării codului în funcție de interfața folosită. Pentru această problemă au venit diferite soluții de reparare a scripturilor de testare, dar pe termen lung prea laborioase și complicate pentru testeri. Atunci această tehnică de automatizare s-a extins spre folosirea de instrumente și framework-uri mai specializate ce necesită cât mai puține modificări ale codului din partea testerilor. Dar și aceste instrumente pot îngreuna procesul de testare dacă nu sunt destul de avansate din punct de vedere tehnologic sau adaptate la interfața folosită. De exemplu, un instrument ce se concentrează pe DOM-ul unei aplicații web nu poate fi folosit pentru o aplicație desktop sau mobil. Aplicațiile mobile vin cu noi elemente de interfață grafică, precum activități, gesturi, interacțiuni dintre elemente, motiv pentru care testarea trebuie adaptată la aceste elemente folosind metodele și tehnologiile potrivite. Adaptarea la noile tehnologii poate fi dificilă pentru că de multe ori un tester nu știe din prima cu ce se confruntă, la cât de repede evoluează tehnologia și drept urmare trebuie să evolueze și tehniciile de testare în mod corespunzător. O altă problemă ce poate interveni este ce metodă de testare acoperă cel mai bine cazurile de testare. De asemenea, unele instrumente moderne scutesc programatorii de pași în plus ce trebuie făcuți cu instrumente mai vechi de testare, motiv pentru care este important să se cunoască noile tehnologii apărute în domeniu.

Mai departe, în ceea ce privește testarea record and playback, poate fi privită ca o tehnică mai avansată de automatizare a testării deoarece înregistrează acțiunile utilizatorului și le replică, dar o problemă ce poate fi întâlnită folosind această tehnică este comiterea de greșeli din partea utilizatorului, în sensul în care multe cazuri de testare importante s-ar putea să nu fie acoperite în fază de recording, ca în cazul testării manuale, cu diferența că partea de playback aduce un plus față de testarea manuală prin automatizarea procesului de testare. Aici intervine tehnica de tes-

tare folosind RPA. Robotii software identifică elementele interfeței grafice folosind computer vision și cazurile de testare sunt create în cadrul platformei acoperind diverse scenarii, apoi sunt primite rapoartele rezultate în urma testării. Fiind o testare bazată pe inteligență artificială în recunoașterea elementelor interfeței grafice, există problema de cât de bine sunt antrenate modelele de inteligență artificială pentru a recunoaște toate elementele corespunzătoare, dar odată cu evolutia tehnologiei și a inteligenței artificiale, și aceste probleme pot fi rezolvate.

Viitoare dezvoltări privind testarea interfeței grafice se pot concentra pe crearea de instrumente destul de avansate astfel încât inteligența artificială să reușească să execute testarea fără niciun ajutor din partea oamenilor, în sensul în care nu doar elementele interfeței grafice să fie recunoscute, dar și cazurile de testare să fie gândite, create și executate în funcție de complexitatea aplicației de către roboți software ce folosesc inteligență artificială.

3. Projy: Aplicație de postare de proiecte

3.1 Analiza aplicației

3.1.1 Descrierea aplicației

Projy este o aplicație de postare de proiecte personale de către persoane din toate domeniile. Aceasta este o aplicație layered, client-server ce are drept scop postarea de proiecte personale, cu scopul final de a fi vizualizate de către recrutorii firmelor ce vor să angajeze persoane ce folosesc anumite tehnologii sau care au o anumită experiență în ceea ce privește diferite aplicații sau proiecte, precum și cu scopul inspirării de idei noi și networking dintre utilizatori. Poate fi privită ca o combinație de aplicații existente în producție, dar cu diferența că înglobează proiecte din toate domeniile și se concentreză exclusiv pe postarea de proiecte.

3.1.2 Funcționalitățile aplicației

Utilizatorul ce folosește aplicația se poate loga folosind username și parola. Dar dacă acesta nu are cont în aplicație, își poate crea un nou cont introducând datele sale și anume cele obligatorii de nume, prenume, username, email și parolă și dacă își dorește, data de naștere, genul și locația. Odată ce acesta se loghează sau își creează cont în aplicație, este redirectionat către pagina principală de feed, unde sunt afișate toate proiectele poste de către utilizatori. Utilizatorul își poate vedea de asemenea pagina personală cu proiectele sale odată ce accesează meniul cu opțiuni. Din meniu, se mai poate alege opțiunea de a posta un proiect, ce odată postat va apărea în feed-ul din pagina principală și pe pagina personală a utilizatorului. Dacă utilizatorul dorește să păstreze doar anumite proiecte în feed, acesta poate să facă swipe pentru ștergerea lor din feedul său. Utilizatorul mai poate să își editeze profilul personal, având opțiunea să își modifice datele personale. De asemenea, acesta își poate șterge contul. Dacă dorește să se logheze cu alt cont, acesta poate alege opțiunea de switch account din meniu, de unde este redirectionat către pagina de login. Aceste cazuri de utilizare ale aplicației se pot observa în diagrama 3.1.

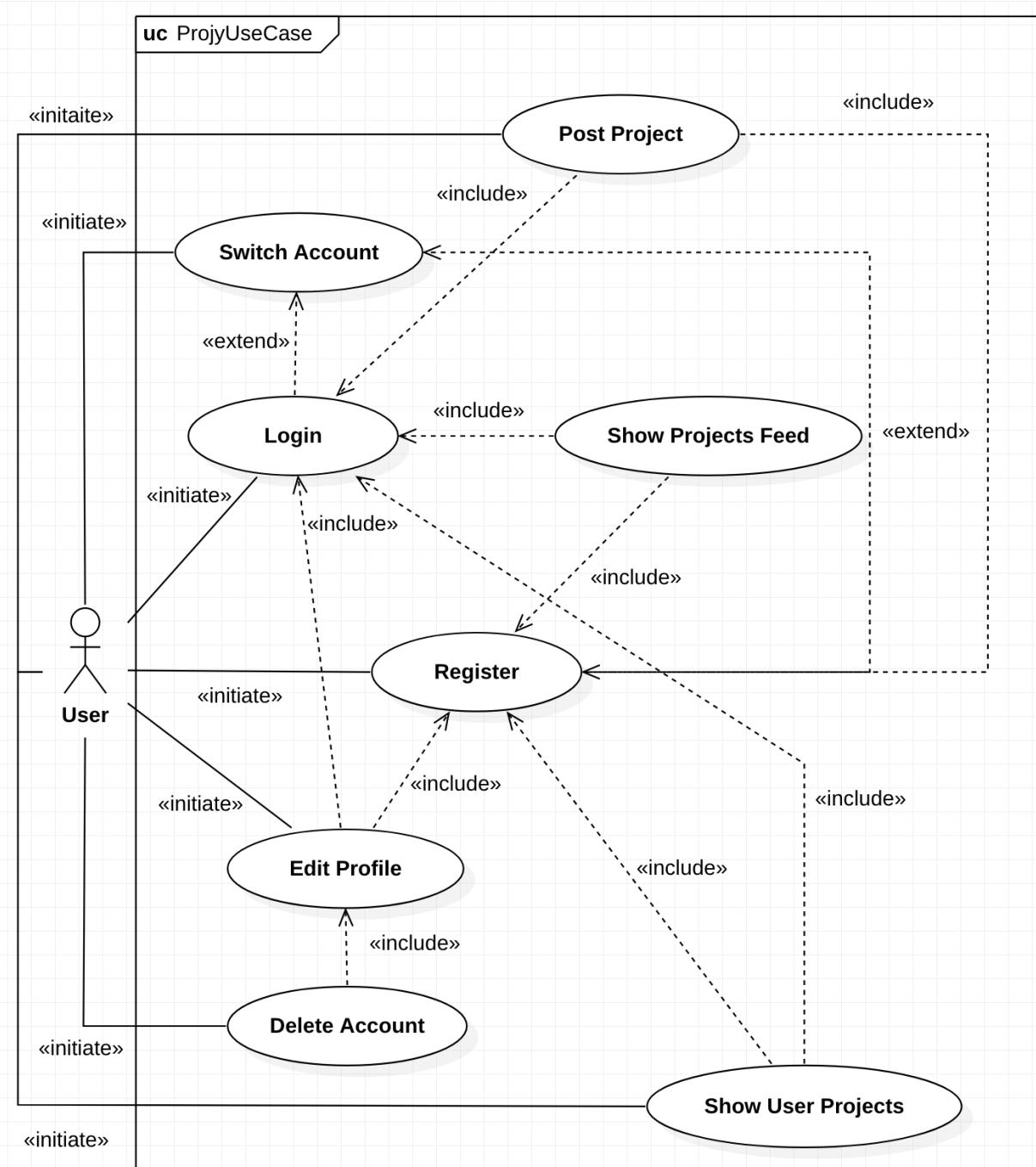


Figura 3.1: Projy UseCase Diagram

3.1.3 Manual de utilizare

Odată ce se deschide aplicația Projy, este afișată pagina de Login. Precum se poate observa în figura 3.2a, utilizatorul se poate loga în aplicație introducând username și parolă ale unui cont existent în field-urile corespunzătoare și ulterior apăsând butonul de Login. Dar dacă acesta nu are deja cont în aplicație, își poate crea un nou cont alegând opțiunea de "Don't have an account? Create one here!". Odată ce

apasă pe link-ul "here", este redirectionat către pagina de înregistrare a noului cont în aplicație, precum este arătat în figura 3.2b. Acesta se poate înregistra introducând datele sale și anume cele obligatorii de nume, prenume, username, email și parolă și dacă își dorește, data de naștere, genul și locația. Toate aceste field-uri se pot completa cu text, iar în field-ul de Birthday se poate alege opțiunea de a alege o dată din calendar. După ce utilizatorul a introdus aceste date și a apăsat butonul de Register, contul de utilizator în aplicație a fost creat.

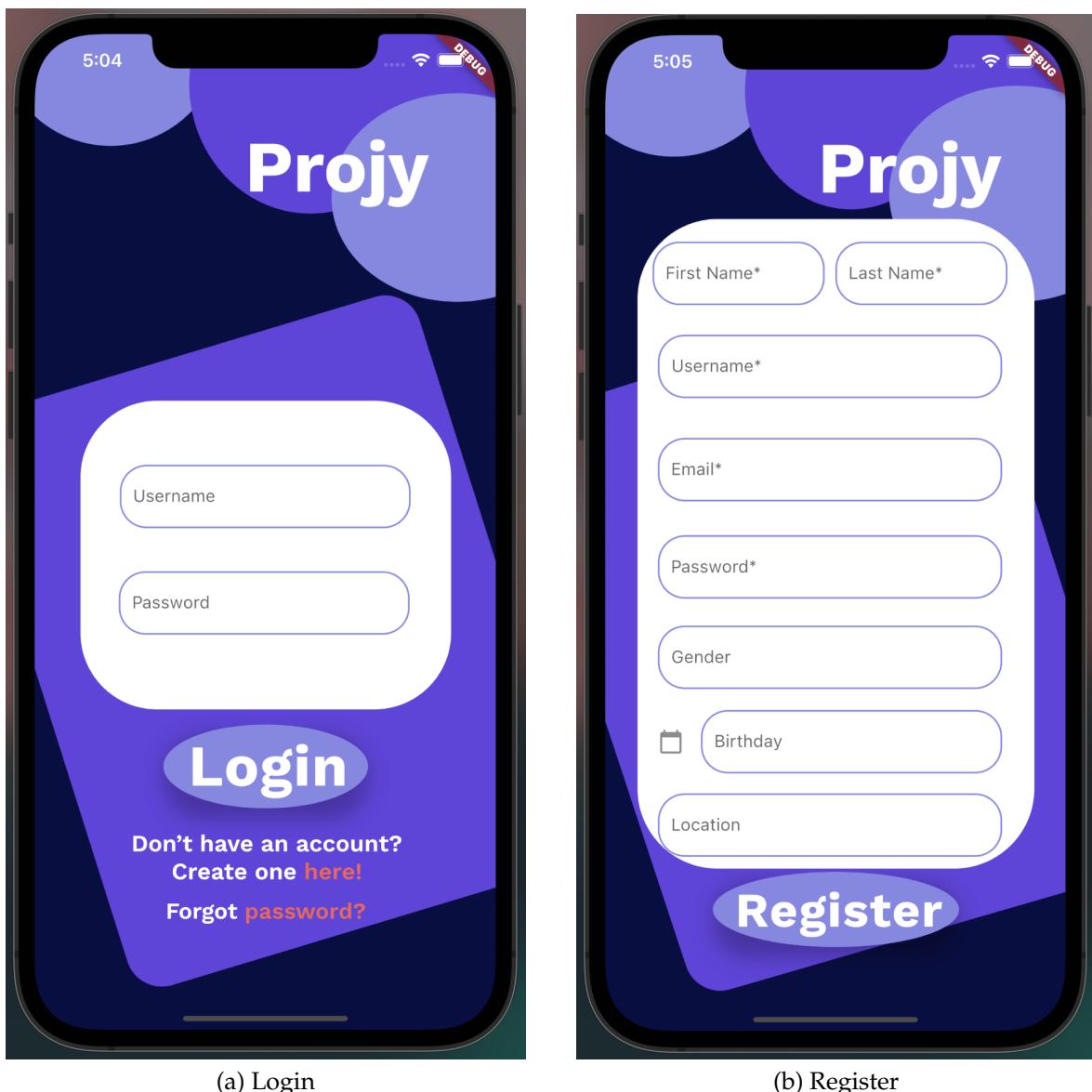


Figura 3.2: Login And Register

După oricare dintre opțiunile alese de către utilizator, Login sau Register, acesta este redirectionat către pagina principală de Home, unde este afișat feed-ul cu proiectele tuturor utilizatorilor din aplicație, arătat în figura 3.3a. Utilizatorul poate deschide meniul cu opțiuni apăsând pe barele de meniu prezente în antetul aplicatiei.

Acestea apar de fiecare dată în același loc pe fiecare pagină a aplicației, pentru ca utilizatorul să poată avea acces la meniu și să execute acțiunile dorite în orice moment. Opțiunile prezente în meniu, precum se poate observa în figura 3.3a, sunt Home, Post Project, Edit Profile și Switch Account. Alegerea opțiunii de Home redirecționează utilizatorul către pagina principală de feed cu proiecte. De asemenea, pentru a ajunge la pagina personală cu proiecte se poate apăsa pe icon-ul de user sau pe username iar utilizatorul va fi redirecționat către pagina sa, afișată în figura 3.4a.

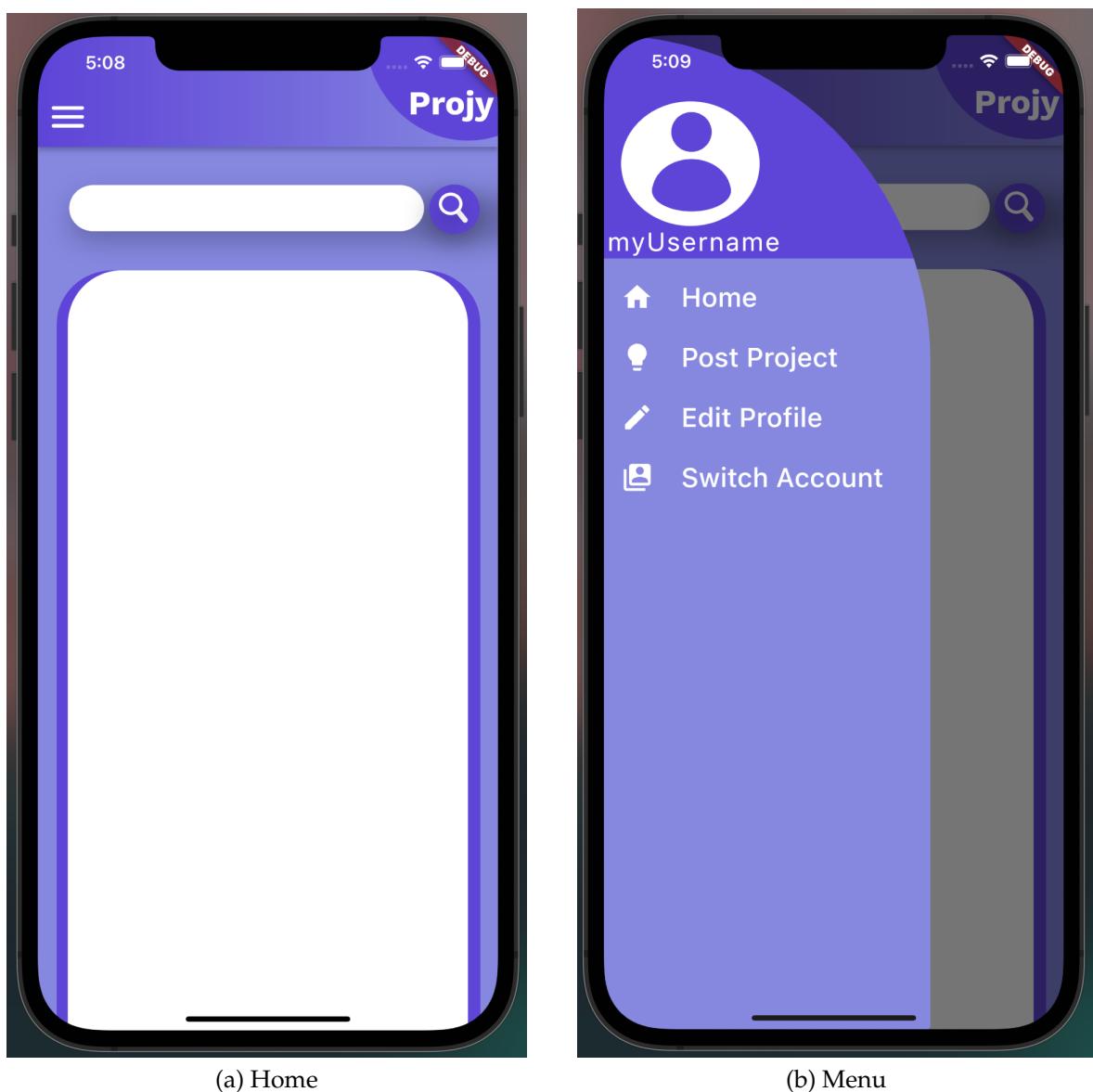
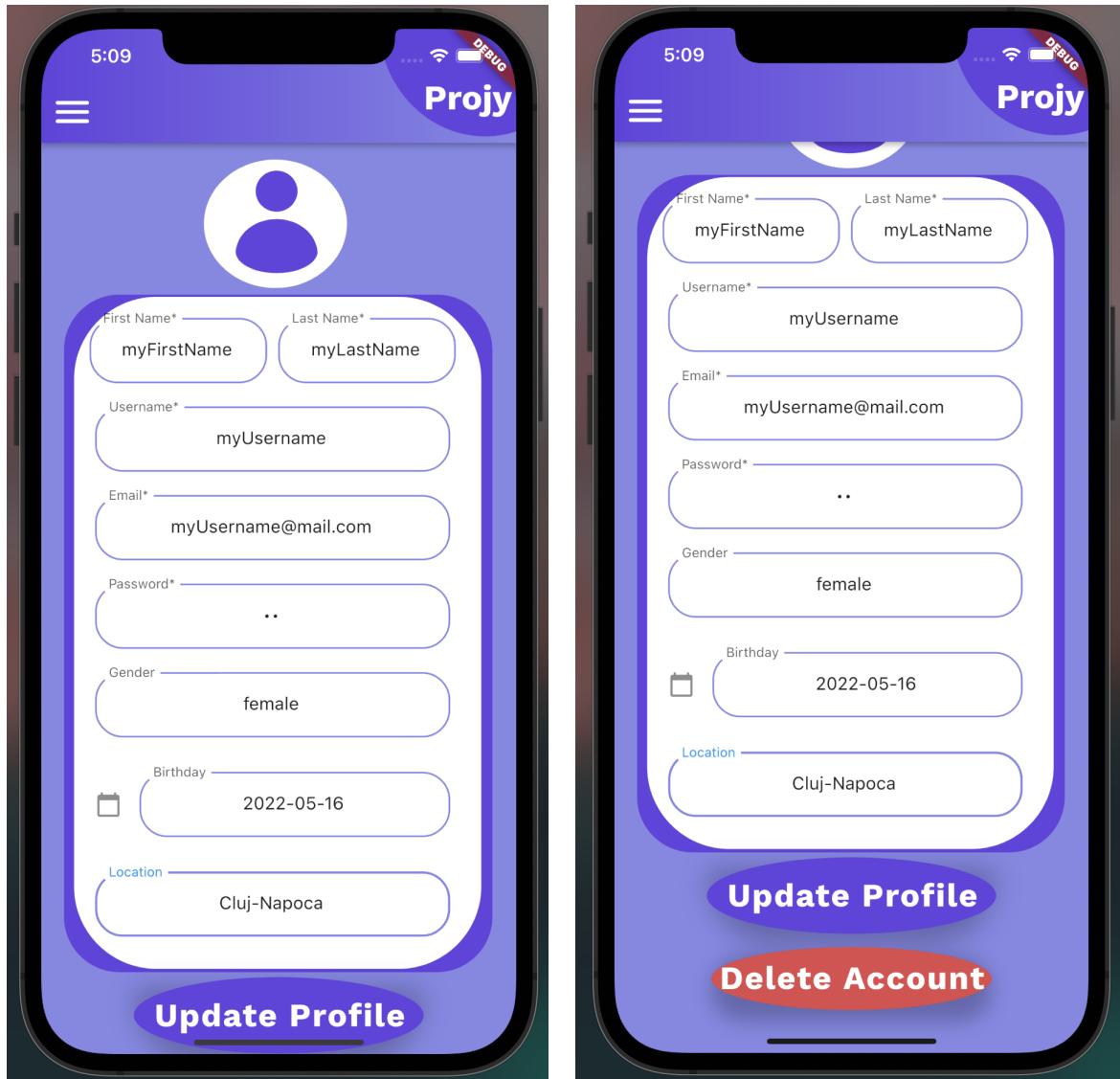


Figura 3.3: Home Page And Menu

Utilizatorul mai poate alege din meniu 3.3b opțiunea de a-și edita detaliile profilului său și va fi redirecționat către pagina afișată în figura 3.4a, unde vor apărea toate detaliile înregistrate de acesta în aplicație și cu opțiunea de a le modifica pe fi-

ecare în parte: primul și al doilea nume, username, email, parolă, gen, data nașterii sau locația. Odată modificate, utilizatorul poate apăsa pe butonul de "Update Profile" și toate detaliile profilului vor fi actualizate în aplicație.



De asemenea, dacă utilizatorul dorește ștergerea contului, acesta poate să scrollă în jos în cadrul acestei pagini pentru apăsarea butonului de "Delete Account", precum se poate observa în figura 3.4b și contul va fi șters din aplicație. Dar dacă utilizatorul dorește să se deconecteze din aplicație fără ștergerea contului, acesta poate alege opțiunea de "Switch Account" din meniu 3.3b și va fi trimis către pagina de Login 3.2a, de unde se va putea loga cu alt cont sau va putea înregistra 3.2b alt cont în aplicație.



Figura 3.4: Home Page And Menu

Dacă utilizatorul alege opțiunea de Post Project din meniu, acesta va fi trimis către pagina de postare de proiecte, precum se poate observa în figura 3.4b. Postarea de proiecte constă în alegerea unui titlu pentru proiect, care poate fi completat în field-ul corespunzător și a unui text corespunzător proiectului său pentru a îl descrie. Odată ce acesta apasă butonul de Post Project, proiectul este postat în aplicație și acesta va putea fi observat de către utilizatori pe pagina principală de feed 3.3a și de asemenea pe pagina personală 3.4a. Pagina principală de feed conține proiectele postate de către toți utilizatorii, iar pagina personală conține doar proiectele utilizatorului logat în aplicație.

3.2 Proiectarea aplicației

3.2.1 Arhitectura client server

Serverul aplicației Projy este format din mai multe pachete ce comunică între ele și anume Database Model, unde sunt prezente entitățile aplicației existente și în baza de date: User și Post, ce moștenesc din BaseEntity, entitatea de bază. Repository este un pachet ce conține UserRepository și PostRepository, corespunzătoare operațiilor de persistență ce se fac în comunicare cu baza de date a aplicației pentru fiecare entitate în parte. Pachetul Services procesează informația primită de la Controller și apelează Repository pentru salvarea informațiilor în baza de date, ce sunt împărțite în clasele UserServiceImpl și PostServiceImpl, corespunzătoare fiecărei entități. Informația pe care Controller o transmite pachetului Services este primită prin apeluri de REST API de la Client și este împărțită la rândul ei în clasele UsersController și PostsController. Aceste clase folosesc obiecte de tip Data Transfer Object (DTO) pentru primirea și transmiterea de informații cu Clientul aplicației. Clientul este format din mai multe pachete ce fac posibilă transmiterea de informații de la interacțiunea utilizatorului cu interfața grafică a aplicației la serverul aplicației. Pachetul de Screens conține toate ecranele (paginile) aplicației la care utilizatorul aplicației are acces: LoginScreen, RegisterScreen, HomeScreen, PersonalPageScreen, UpdateProfileScreen și PostProjectScreen. Prin intermediul butoanelor, field-urilor de text și diferitelor elemente grafice prezente pe ecrane, utilizatorul își comunică acțiunile și datele ce urmează a fi transmise și procesate mai departe către celelalte pachete ale Clientului. Repository preia aceste informații și le împarte în funcție de entitățile corespunzătoare în UserRepository și PostRepository, pe care mai departe le transformă în entitățile User și Post din Model, ce urmează a fi salvate în baza de date locală a aplicației, ce conține tabelele UserEntry pentru toți utilizatorii ce au cont în aplicație, LoggedUserEntry pentru utilizatorul logat local în aplicație și PostEntry pentru toate postările (proiectele) utilizatorilor. Controller face comunicarea dintre Client și Server prin Hypertext Transfer Protocol (HTTP) în HttpHelper, folosindu-se apeluri de REST API pentru transmiterea în format JSON a entităților ce urmează a fi procesate de către Client și Server. WebSocketHelper din Controller facilitează procesarea datelor cu ajutorul WebSocket-urilor ce actualizează în timp real feed-ul pentru clienți, fiecare Client fiind corespunzător unui utilizator ce folosește aplicația, indiferent de dispozitivul folosit sau contul folosit. Astfel, Clientul comunică cu Serverul pentru funcționarea facilă a aplicației și salvarea datelor global și local ce urmează a fi folosite în aplicație de către utilizatori, precum se poate observa în arhitectura Client Server ilustrată în diagrama 3.5.

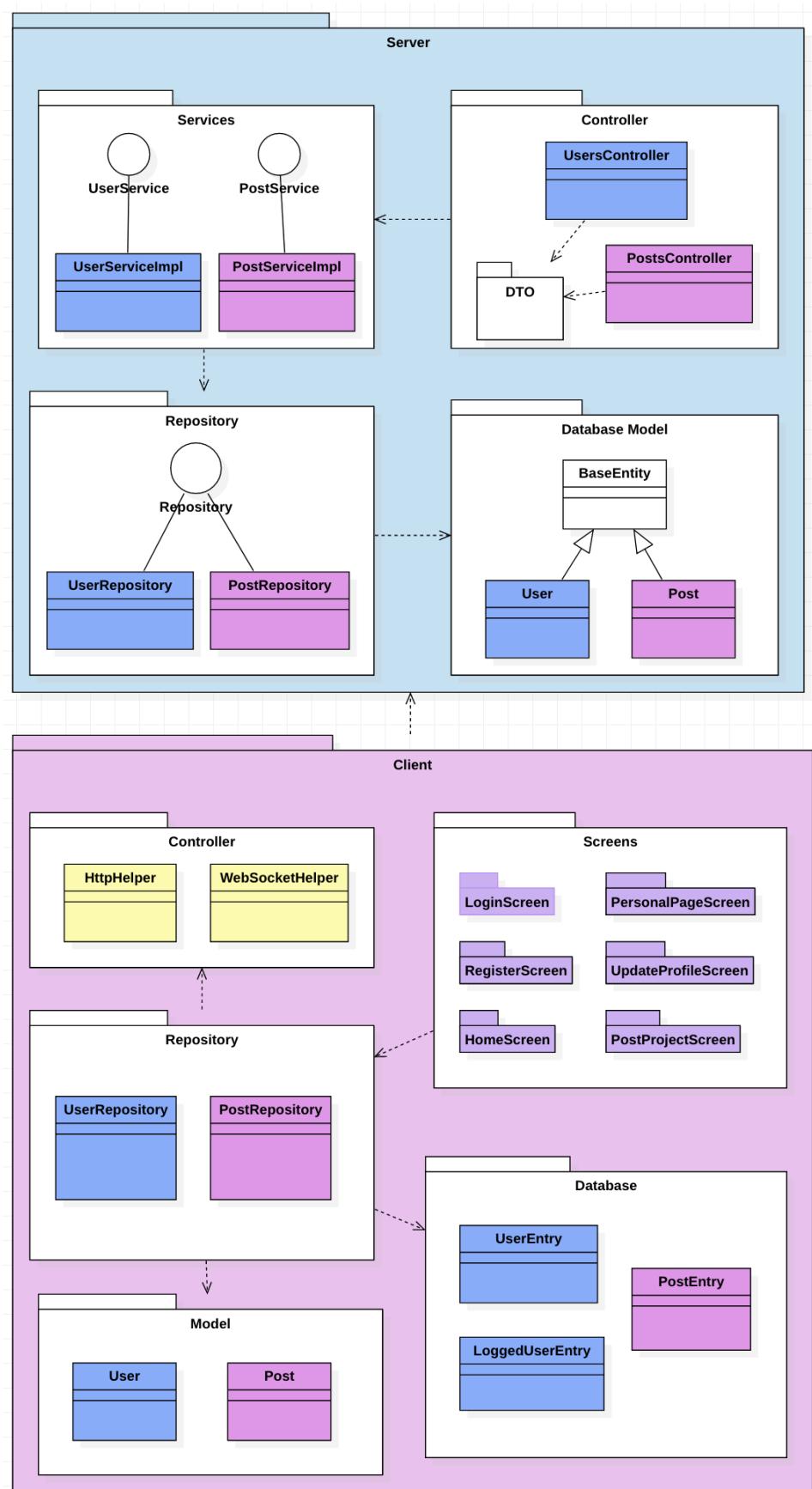


Figura 3.5: Arhitectura Client Server

3.2.2 Persistența datelor în baza de date

Datele folosite în aplicație sunt persistate într-o bază de date relațională globală de pe server și o bază de date locală de pe client. Baza de date globală are drept tabelă User pentru salvarea datelor utilizatorilor înregistrati în aplicație și anume id - cheie primară, first name, last name, username, email, password, birthday, gender și location. Aceasta este într-o relație one-to-many cu tabela Post, ce reține toate proiectele poste de către utilizatori. Post conține drept attribute id - cheie primară, title, description, date și user id - cheie străină corespunzătoare atributului id din tabela User. Această relație one-to-many este datorată faptului că un utilizator poate posta mai multe proiecte, dar aceeași postare nu poate fi făcută de mai mulți utilizatori. În timp ce tabelele prezente în diagrama 3.6 corespund bazei de date globale, baza de date locală mai are în plus o tabelă LoggedUser 3.7 ce reține datele utilizatorului curent logat în aplicație. Baza de date locală are drept scop reținerea informațiilor introduse local de către utilizatori, pentru ca în eventualitatea pierderii conexiunii la server sau la rețeaua de internet, datele introduse să nu se piardă, ci să fie salvate local ca în momentul revenirii conexiunii să fie folosite.

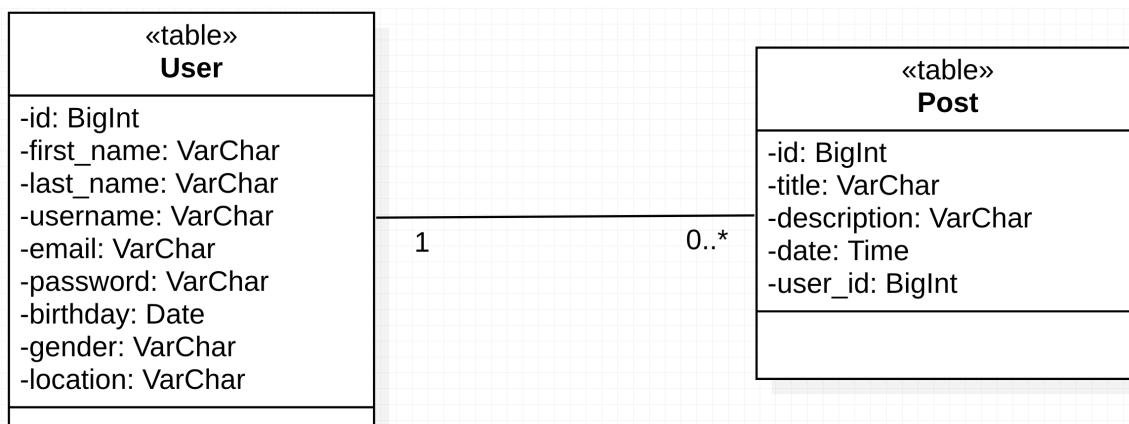


Figura 3.6: Baza de date

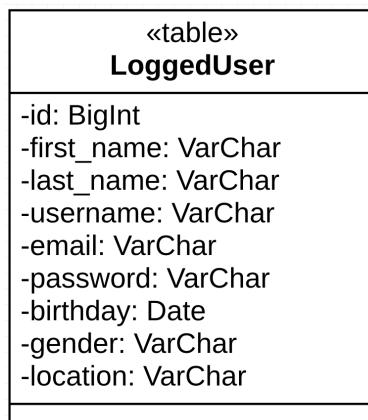


Figura 3.7: Local

3.3 Implementarea aplicației

3.3.1 Arhitectura claselor aplicației

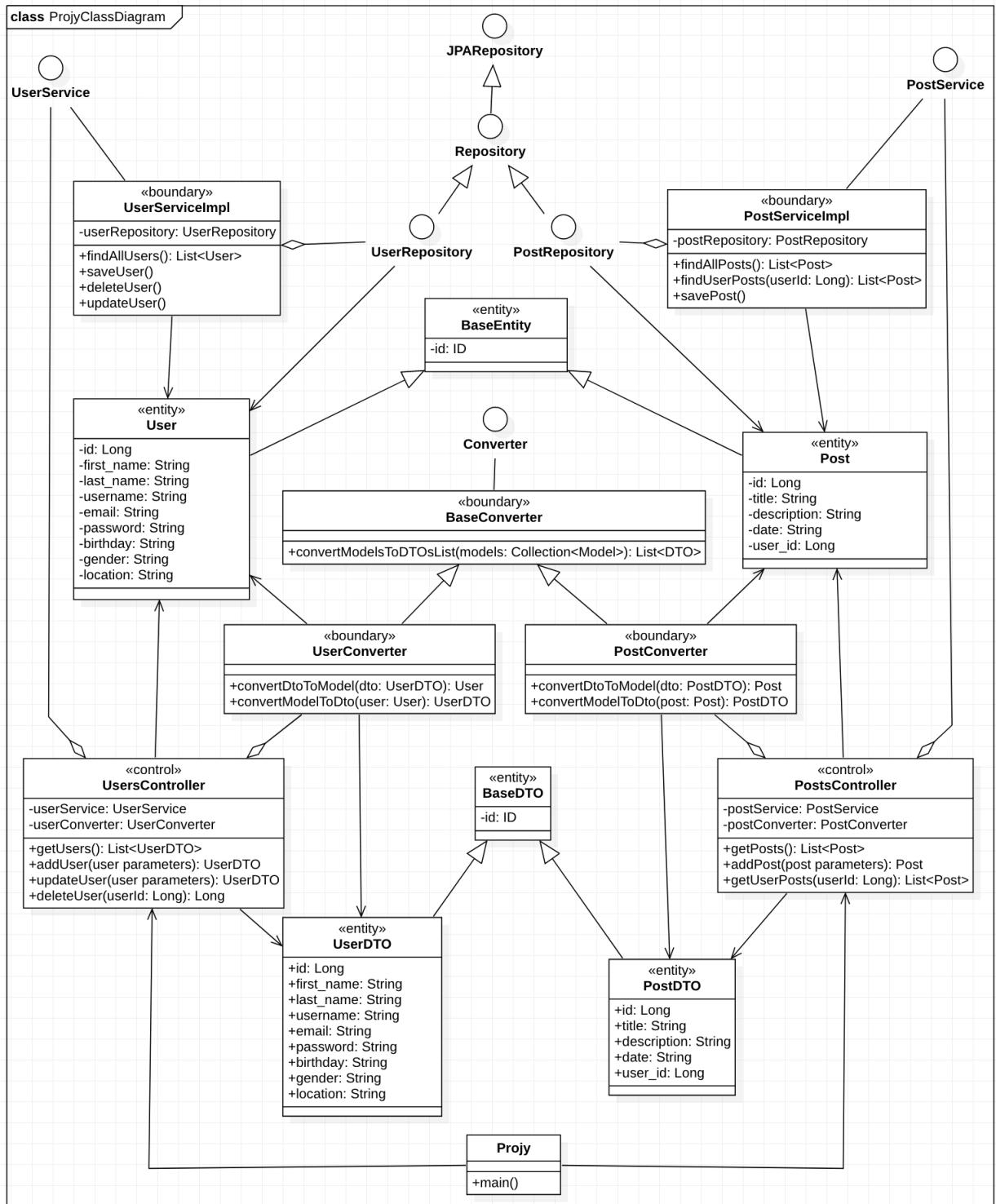


Figura 3.8: Diagrama de clase

Diagrama de clase a aplicației, arătată în figura 3.8, evidențiază aspecte legate de reducerea codului scris prin moștenirea de JPARepository 3.12 de către clasele de Repository corespunzătoare fiecărei entități (UserRepository și PostRepository) și transmiterea metodelor CRUD acestora. Clasele de Service UserServiceImpl și PostServiceImpl implementează aceste metode împreună cu metodele de filtrare corespunzătoare. Toate aceste clase folosesc entitățile de User și Post ce moștenesc dintr-o entitate de bază și beneficiază la rândul lor de funcționalități deja implementate datorită librăriei Lombok 3.11. Clasele UsersController și PostsController folosesc clasele de Service pentru apelarea metodelor corespunzătoare și comunică prin REST API cu clientul pentru a transmite și primi datele necesare prin metodele HTTP de GET, POST, PUT și DELETE. Pentru transmiterea entităților din model către client, este nevoie de o convertire a lor la un obiect de tipul Data Transfer Object (DTO) și invers când se primesc informații în format JSON de la client, deoarece este mai eficient și mai securizat ca doar anumite date să fie transmise către client. Astfel, clasele UserConverter și PostConverter care moștenesc din clasa BaseConverter ce implementează Converter, execută această transformare dintre model și DTO și invers. Pentru aceasta sunt folosite entitățile UserDto și PostDto ce moștenesc din BaseDTO. În final, clasa Main Projy rulează tot conținutul aplicației de server. Structura clientului este asemănătoare cu cea a serverului, precum se poate observa în diagrama arhitecturii client server 3.5, cu diferența că există ecranele interfeței grafice ce preiau datele introduse de către utilizatori și baza de date pezintă și o tabelă ce reține datele utilizatorului logat în aplicație.

3.3.2 Tehnologii folosite în implementarea aplicației

Clientul și serverul aplicației sunt conectate prin apeluri de REST API dintre un client implementat în Flutter și un server implementat în Java Spring Boot. Am folosit Flutter [Flu22b] pentru implementarea clientului, care este un framework creat de Google ce folosește Dart [Dar22] ca limbaj de programare și oferă avantajul de a putea rula codul scris pentru interfață grafică pe toate tipurile de dispozitive: Android, iOS, Linux, macOS, Windows, Google Fuchsia și Web. Pentru partea de UI Design a aplicației am folosit Figma [Fig22] pentru conceperea designului și FlutLab [Flu22a] pentru convertirea designului interfeței grafice la codul în Flutter. De asemenea, salvarea într-o bază de date locală a entităților aplicației este facilitată de Moor [Pub22b], o librărie ce generează codul pentru operațiile bazei de date relaționale SQLite [SqL22]. Un avantaj pe care îl oferă această tehnologie este actualizarea în timp real a datelor din baza de date locală pe interfața grafică a aplicației prin folosirea de stream-uri. Datele din stream-uri sunt primite în timp real din baza de date prin apelul funcției watch(), precum se poate observa în figura 3.9.

```

class Bloc {
    late final Stream<List<UserEntry>> _users;
    late final Stream<List<PostEntry>> _posts;
    late final Stream<List<LoggedUserEntry>> _logged_user;
    late PostEntry _post;
    Stream<List<PostEntry>> get homeScreenEntries => _posts;
    Stream<List<LoggedUserEntry>> get loggedUserEntry => _logged_user;

    final Connectivity _connectivity = Connectivity();
    StreamSubscription<ConnectivityResult>? _connectivitySubscription;

    late HttpHelper h = HttpHelper();
    var logger = Logger();
    final Database db;

    Bloc() : db = Database() {
        _posts = db.watchPosts();
        _users = db.watchUsers();
        _logged_user = db.watchLoggedUser();
        updateUsersDB();
        updatePostsDB();
    }

    Future<void> _updatePostConnectionStatus(
        ConnectivityResult result, LocalPostsCompanion post) async {
        switch (result) {
            case ConnectivityResult.wifi:
            case ConnectivityResult.mobile:
                {
                    logger.d('status: online... adding: ' + post.toString());
                    Post mypost = Post(
                        id: '',
                        title: post.title.value,
                        description: post.description.value,
                        date: post.date.value,
                        userid: post.userid.value);

                    _logged_user.first.then((value) {
                        h.addPost(mypost, value.first.id).then((value) {
                            if (value != null) {
                                logger.d('Added post to server: ' + mypost.toString());
                            }
                        });
                    });
                }
        }
    }
}

```

Figura 3.9: Flutter Streams

În funcție de conexiunea la rețeaua de internet, identificată prin folosirea librăriei Connectivity [Pub22a] și subscrierea la un stream de conectivitate *StreamSubscription < ConnectivityResult >*, se face apelul de REST de pe server și se salvează local datele ce urmează să fie afișate în timp real pe interfață grafică. Dacă nu există conexiune la internet, datele se salvează doar local și când revine conexiunea se preiau din baza de date locală și se adaugă și pe server. Modul în care aceste date sunt afișate în timp real pe interfață grafică este determinat de un *ListView.builder()* ce primește stream-ul respectiv pentru a-l afișa, precum este arătat în figura 3.10.

```

class _UsersListWidgetState extends State<UsersListWidget> {
  Bloc get bloc => Provider.of<Bloc>(context, listen: false);

  @override
  Widget build(BuildContext context) {
    return ListView.builder(
      itemCount: widget.posts.length,
      itemBuilder: (context, index) {
        final post = PostEntry(
          id: widget.posts[index].id,
          title: widget.posts[index].title,
          description: widget.posts[index].description,
          date: widget.posts[index].date,
          userid: widget.posts[index].userid); // PostEntry

        final mypost = Post(
          id: widget.posts[index].id.toString(),
          title: widget.posts[index].title,
          description: widget.posts[index].description,
          date: widget.posts[index].date,
          userid: widget.posts[index].userid); // Post

        final strpost = mypost.toString();
      }
    );
  }
}

```

Figura 3.10: Flutter Builder

Spring Boot [Spr] este un framework ce folosește Java [Jav22] ca limbaj de programare și cu ajutorul acestui framework, împreună cu instrumentul Gradle [Gra22] asociat am implementat serverul aplicației. Baza de date a serverului a fost construită pe baza tehnologiei Java Persistence API (JPA) [Spr22], o tehnologie din categoria Object–relational mapping (ORM), ce persistă datele dintre entitățile aplicației și baza de date relațională MySQL [MyS22]. În figura 3.11 se pot observa adnotările folosite pentru maparea entităților din modelul aplicației la tabelele din baza de date și relațiile dintre tabele și anume @OneToMany pentru relația one-to-many de la User la Post. Un alt avantaj al folosirii adnotărilor în Spring Boot este evitarea scrierii de cod repetitiv, precum este în cazul librăriei Lombok [Lom22], ce oferă adnotări precum @Getter și @Setter, care înlocuiesc codul de get și set al entităților, @EqualsAndHashCode pentru egalitatea dintre entități, @AllArgsConstructor și @NoArgsConstructor pentru constructorii clasei și @ToString pentru forma de String a entității. De asemenea, fiecare tip de clasă prezintă o adnotare corespunzătoare și anume @Entity pentru entitățile din model, @Repository, @Service și @Controller. De asemenea, @NamedEntityGraph preia din baza de date în mod eficient doar graful corespunzător relației dintre User și Post, fără a prelua toate relațiile și tabelele din baza de date ce nu au legătură cu operația cerută, preluarea fiind de tip Lazy, în

sensul în care se preia doar la momentul la care este nevoie.

```

@NamedEntityGraphs({
    @NamedEntityGraph(name = "userWithPosts",
        attributeNodes = @NamedAttributeNode(value = "posts"))
})
@Entity
@EqualsAndHashCode
@NoArgsConstructor @AllArgsConstructor
@Data @Builder
@ToString
public class User extends BaseEntity<Long> {
    @Column(unique = true)
    private String username;
    private String password;
    private String firstName;
    private String lastName;
    private String email;
    private String location;
    private String gender;
    private String birthday;

    @JsonManagedReference
    @OneToMany(mappedBy = "user", cascade = CascadeType.ALL, fetch = FetchType.LAZY)
    private List<Post> posts = new ArrayList<>();

    @JsonManagedReference
    @OneToMany(mappedBy = "user", cascade = CascadeType.ALL, fetch = FetchType.LAZY)
    private List<Comment> comments = new ArrayList<>();

    @JsonManagedReference
    @OneToMany(mappedBy = "fromUser", cascade = CascadeType.ALL, fetch = FetchType.LAZY)
    private List<Message> messagesFrom = new ArrayList<>();

    @JsonManagedReference
    @OneToMany(mappedBy = "toUser", cascade = CascadeType.ALL, fetch = FetchType.LAZY)
    private List<Message> messagesTo = new ArrayList<>();
}

```

Figura 3.11: Spring Boot Adnotations

```

@NoArgsConstructor
@RequiredArgsConstructor
public interface SocialRepository <T extends BaseEntity<ID>, ID extends Serializable> extends JpaRepository<T, ID> {
}

```

Figura 3.12: Spring Boot JPA

În figura 3.12 se poate observa cum moștenirea clasei JpaRepository de către Repository rezultă în moștenirea metodelor de Create, Read, Update, Delete (CRUD), fără ca acestea să mai fie scrise de în cod.

3.3.3 Arhitectura interfețelor web și mobil

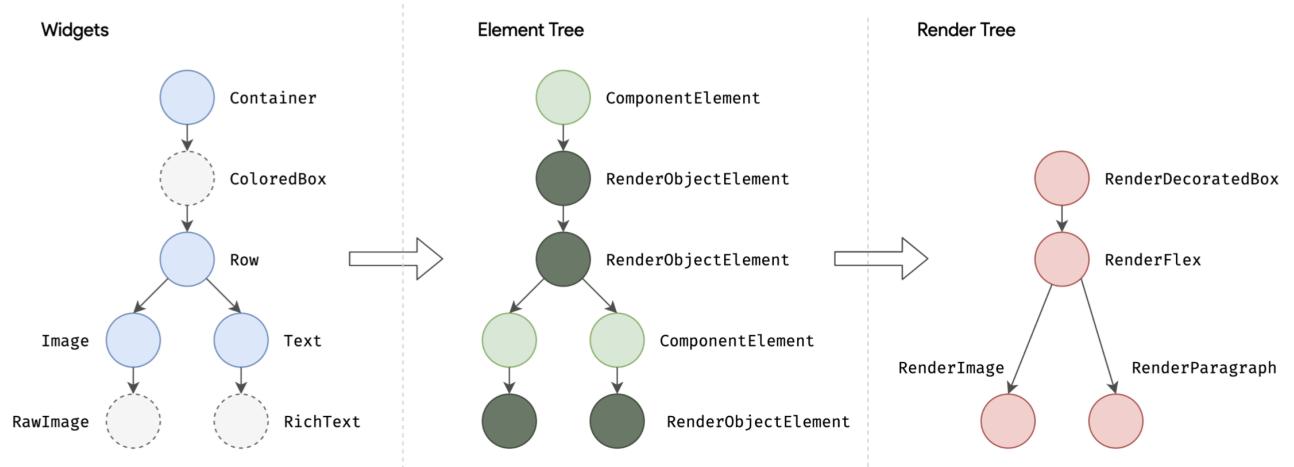


Figura 3.13: Arhitectura Flutter [Flu22c]

Toate interfețele web și mobil - IOS și Android sunt rezultatul aceluiași cod scris în Flutter. Cu toate că fiecare tip de interfață prezintă câte o arhitectură diferită, precum am cercetat în capitolele teoretice anterioare (interfața web are la bază elemente de tip DOM și interfețele mobil prezintă o arhitectură event-based), Flutter reușește să transforme codul scris în elemente specifice fiecărui tip de interfață, existând diferențe între modul în care sunt afișate în funcție de scalabilitatea și dimensiunea elementelor grafice ale interfeței grafice. După cum se poate observa în figura 3.13, majoritatea widget-urilor din Flutter sunt redate de un obiect care moștenește din subclasa `RenderBox`, ce reprezintă un `RenderObject` de dimensiune fixă într-un spațiu cartezian 2D.

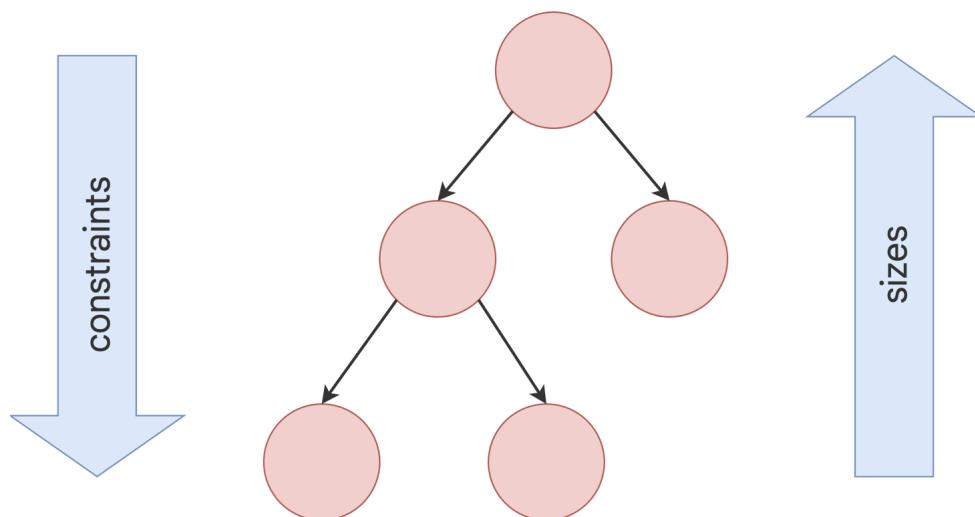


Figura 3.14: Randarea elementelor în Flutter [Flu22c]

`RenderBox` oferă baza unui model de constrângere de tip casetă, stabilind o

lătime și o înălțime minimă și maximă pentru fiecare widget care urmează să fie randat. Pentru a rezulta aspectul interfeței grafice, Flutter parcurge arborele de randare într-o traversare în profunzime și transmite constrângerile de dimensiune de la părinte la copil. În determinarea mărimei acestuia, copilul trebuie să respecte constrângerile care îi sunt date de părintele său. Nodurile copil răspund transmitând o dimensiune obiectului lor părinte în limitele constrângerilor stabilite de nodul părinte, precum se poate observa în figura 3.14.

3.4 Testarea aplicației

Există diferite metode de testare a interfețelor grafice și anume: Record and Replay (înregistrarea acțiunilor unui utilizator și repetarea lor), folosind Robotic Process Automation (există diferite moduri în care poate fi folosit pentru testare, printre care record actions, programatic sau folosind computer vision), prin integration testing în Flutter, prin testare automată sau prin script-uri, folosind diferite framework-uri (de exemplu Selenium). Fiecare prezintă rezultate diferite, dar și folosirea lor diferă din punct de vedere al eficienței atunci când este utilizată de programatori.

3.4.1 Framework-uri folosite în testarea interfeței grafice

Datele folosite pentru testarea aplicației sunt interfețele grafice de pe mai multe tipuri de dispozitive: mobil și web. De asemenea, aceste interfețe grafice sunt formate din mai multe componente: ecrane diferite, împreună cu navigarea dintre ele, diferențele funcționalități care trebuie să funcționeze conform așteptărilor și funcționarea corectă a widget-urilor și elementelor grafice ale aplicației. Pentru a testa aplicația există 3 tipuri de testare în Flutter: unit testing, widget testing și integration testing, iar ultima dintre ele este cea care testează interfața grafică a aplicației.

Un prim tip de testare a interfeței grafice a aplicației a fost testarea manuală, în care am parcurs toate scenariile de folosire a aplicației manual, astfel încât să pot aduce modificări aplicației înaintea folosirii testării automate și scripturile de testare să nu fie nevoie să fie reparate odată cu schimbarea interfeței grafice.

În cele ce urmează, am testat interfața grafică a aplicației Projy pe interfața mobilă IOS folosind testarea automată prin integration testing din Flutter [Flu22d]. Am testat aplicația Projy pe un set de date în care flow-ul este cel de identificare, click și completare a field-urilor corespunzătoare cu username și parolă și login în aplicație prin apăsarea butonului corespunzător. Prin aceste acțiuni se verifică existența ecranului de Login, a field-urilor și butoanelor respective și ajungerea la pagina principală de Home după logarea în aplicație. Toate testele au avut succes și un timp

minim de 600ms. În timpul folosirii a integration testing sunt simulate acțiunile utilizatorului pe interfață grafică pentru a verifica dacă fiecare caz ajunge la rezultatul așteptat. Această tehnică de testare face parte din categoria testării automate, folosindu-se scripturi de testare scrise în cod, precum se poate observa în figura 3.15.

```
void main() {
    IntegrationTestWidgetsFlutterBinding.ensureInitialized();

    Run | Debug
    testWidgets(
        "Login screen -> complete Username, Password fields -> press Login Button -> Intro screen ",
        (WidgetTester tester) async {
            app.main();

            await tester.pump();
            expect(find.byType(GeneratedLoginscreenWidget), findsOneWidget);

            Finder usernameField = find.byKey(const Key('Username'));
            expect(usernameField, findsOneWidget);
            await tester.tap(usernameField);
            await tester.enterText(usernameField, 'b');

            Finder passwordField = find.byKey(const Key('Password'));
            expect(passwordField, findsOneWidget);
            await tester.tap(passwordField);
            await tester.enterText(passwordField, 'b');

            final loginButton = find.byKey(const Key('Login'));
            expect(loginButton, findsOneWidget);
            await tester.tap(loginButton);
            await tester.pumpAndSettle(const Duration(seconds: 2));

            expect(find.byType(GeneratedHomescreenWidget), findsOneWidget);
        });
}
```

Figura 3.15: Integration Testing in Flutter

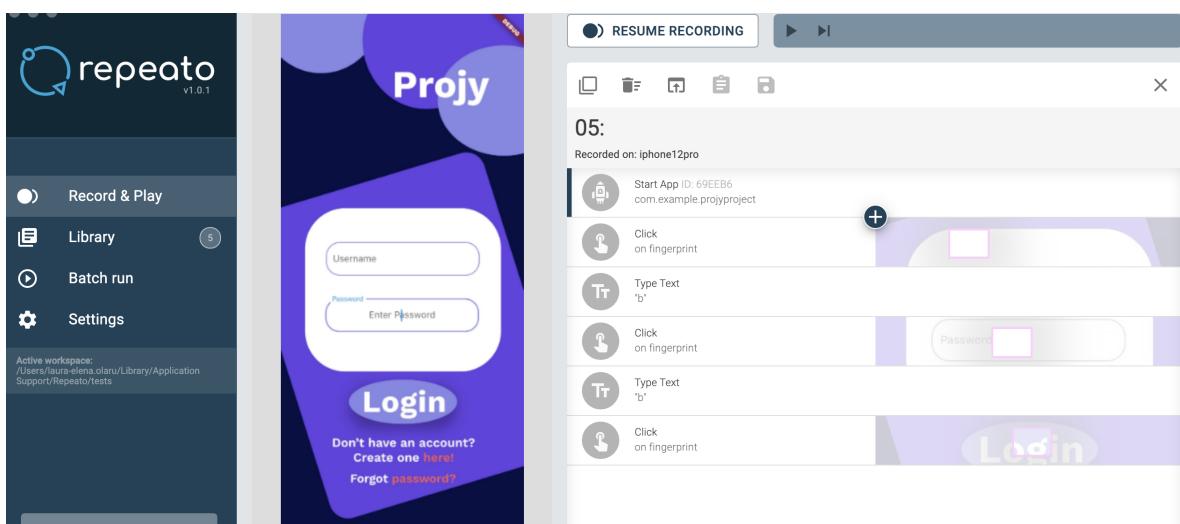


Figura 3.16: Record and Replay IOS testing using Repeato

În continuare, am testat aplicația Projy folosind testarea Record and Replay prima dată pe interfața mobilă IOS și apoi pe interfața web Chrome, Flutter oferind suport pentru ambele tipuri de interfețe.

În ceea ce privește interfața mobilă IOS, am folosit aplicația Repeato [Rep22] de testare Record and Replay. Aceasta reușește să identifice elementele interfeței grafice, oferind capturi de ecran ale acestora și asociindu-le cu acțiunile corespunzătoare în urma înregistrării lor, precum click și type text, cum este afișat în figura 3.16. Beneficiile acestui framework sunt posibilitatea salvării cazurilor de testare într-un batch unde pot fi rulate la unison, dar și rularea lor separată în urma înregistrării acțiunilor pe interfața grafică. De asemenea, este un instrument compatibil cu Flutter, ce permite conexiunea la un simulator de IOS al aplicației.

Pe altă parte, pentru testarea interfeței web Chrome a aplicației Projy am folosit framework-ul Selenium IDE [Sel22b], ce permite atașarea acestuia în browser și identificarea elementelor în urma înregistrării acțiunilor de pe interfața grafică, precum se poate observa în figura 3.18. Flow-ul acțiunilor din browser este cel de alegerea opțiunii de a înregistra un nou cont prin click, redirectionare spre pagina de Register și completarea field-urilor corespunzătoare. Acțiunile sunt identificate și salvate în urma înregistrării lor în aplicație și apoi sunt derulate din nou pentru testarea cu succes, precum se poate observa în figura 3.17.

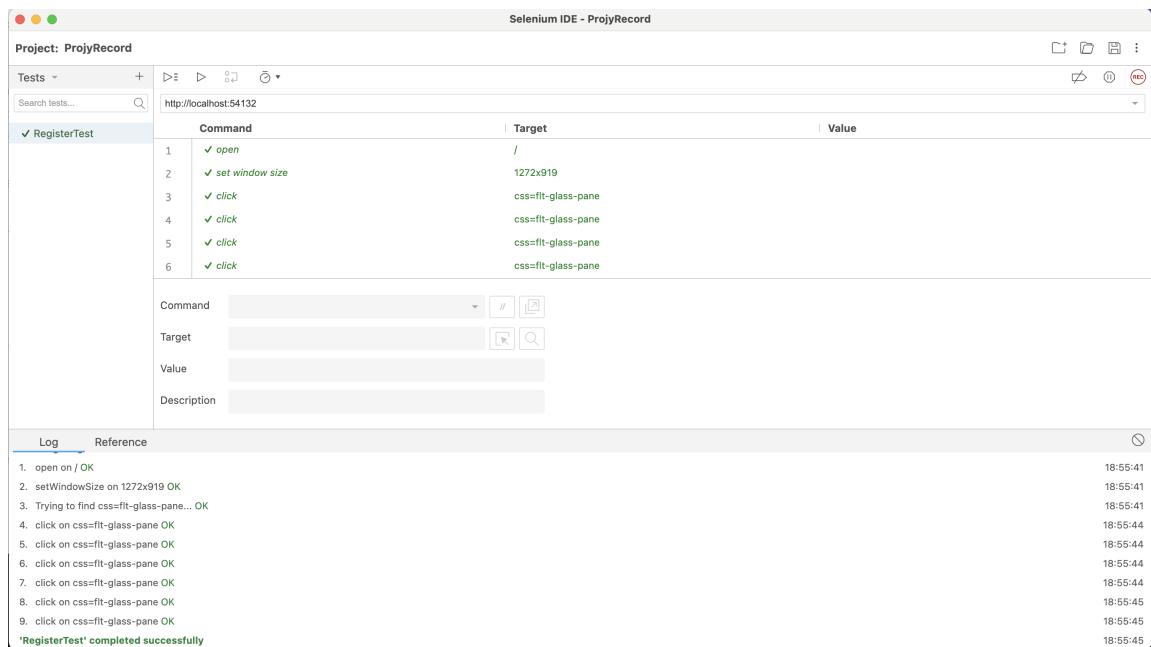


Figura 3.17: Record and Replay Chrome testing using Selenium IDE

Făcând o paralelă între testarea interfeței mobile IOS și testarea interfeței web Chrome, Chrome este mai eficient, timpul necesar de testare fiind mai scurt decât în cazul IOS. De asemenea, Selenium IDE este un framework mult mai potrivit pentru testare, deoarece, cu toate că Repeato oferă feature-uri de afișare a capturilor

de ecran și a modului în care funcționează aplicația în momentul înregistrării, Selenium IDE acționează direct în browser fără a cauza încetinirea testării și oferă un raport complet de parcursere a cazurilor de testare în urma înregistrării și simulării acțiunilor prin replay 3.17.

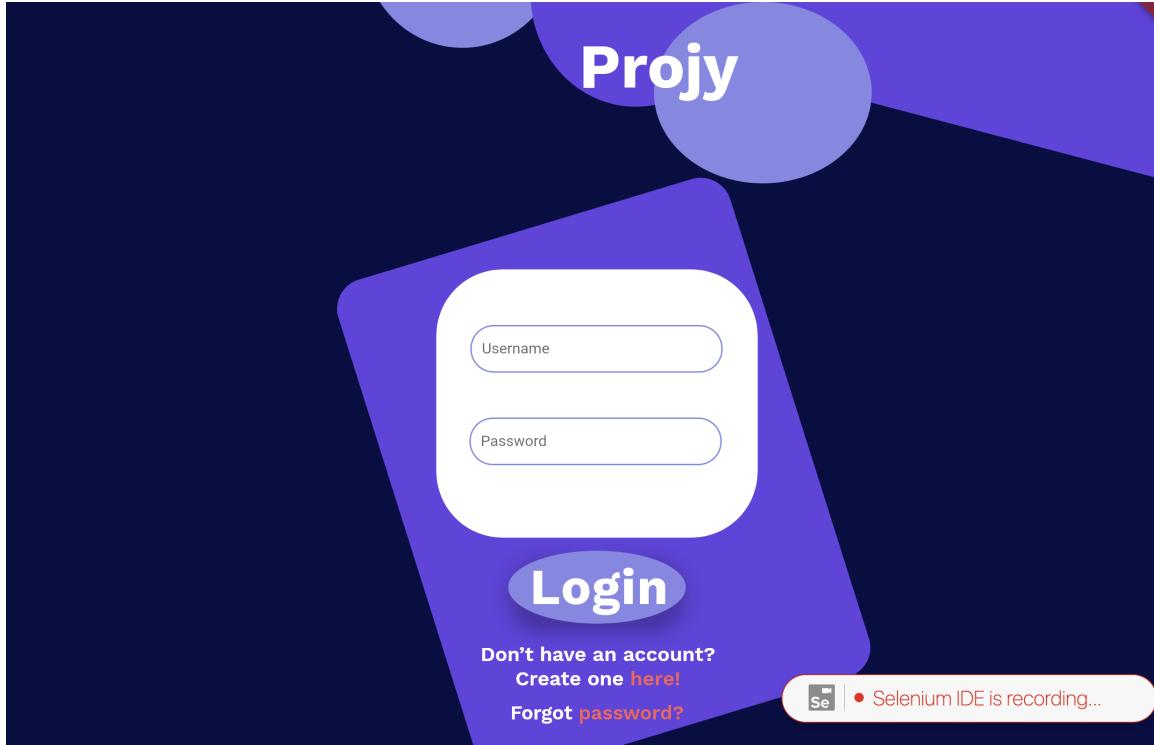


Figura 3.18: Recording Chrome in testing using Selenium IDE

Comparativ cu celelalte tipuri de testare, testarea Record and Replay este cea mai rapidă și cel mai ușor de folosit. Cu toate acestea, testarea automată acoperă mai multe cazuri și este mai eficientă în identificarea cazurilor de excepție, fiind mai integrată în codul aplicației și poate evidenția mai ușor erorile din aplicație. Testarea manuală de asemenea poate acoperi mai multe cazuri de testare, dar testarea automată se execută mai rapid, deci poate găsi mai ușor vulnerabilități în sistem.

3.5 Comparații cu aplicații similare

Projy este o aplicație de postare de proiecte de către utilizatori din toate domeniile, cu scopul creșterii profesionale ale utilizatorilor. Există aplicații similare cu aceasta, precum GitHub [Git22], aplicație de postare de proiecte din domeniul programării, sau Linkedin [Lin22], aplicație de networking din domeniul profesional. Cu toate că sunt aplicații similare, scopul aplicației Projy este de a conecta utilizatori din toate domeniile prin postarea de proiecte personale, fiind o platformă destinată exclusiv proiectelor, ce pot avea potential pentru alți utilizatori ai aplicației, la nive-

Iul oferirii de oferte de job în funcție de abilitățile dovedite în facerea proiectelor sau la nivelul inspirării de idei noi și networking dintre utilizatori.

Posibile continuări ale aplicației ar putea fi funcționalitatea de follow dintre utilizatori pentru filtrarea proiectelor anumitor utilizatori, includerea unui chat pentru conversațiile dintre utilizatori și activarea notificărilor pentru proiecte noi apărute.

4. Concluzii

Concluzionând, în urma realizării lucrării, am deprins abilități de cercetare pe care le-am folosit în studierea domeniului testării interfeței grafice, descoperind diferite perspective din domeniu, precum importanța testării, ce înseamnă fiecare tip de testare în parte, care sunt avantajele și dezavantajele fiecărui tip de testare, de ce concepte și metodologii este nevoie pentru studierea lor și ce framework-uri și instrumente sunt necesare pentru fiecare tip de testare. Tipurile de testare studiate au fost testarea manuală, automatizarea testării folosind scripturi, testarea automată, testarea folosind Record and Playback și testarea folosind Robotic Process Automation. În ceea ce privește testarea manuală, am descoperit diferite metodologii de testare ce constau în tehnici bazate pe experiența testerilor și tehnici bazate pe stările unui automat finit construit în urma interacțiunii cu interfața grafică. Am descoperit cum se automatizează testarea folosind scripturi și în ce constau diferențele metodologii pentru repararea scripturilor de testare în urma apariției schimbărilor pe interfața grafică, metodologii precum REST, SITAR, ATOM și CHATEM. De asemenea, am observat ce impact are schimbarea interfeței grafice odată cu schimbarea tipurilor de interfață desktop, web și mobil. Am parcurs diferențele framework-uri și tehnologii de testare folosind scripturi, odată cu evoluția lor în timp și ce caracteristici prezintă fiecare din ele, printre care se numără Abbot Costello, Selenium și Cypress. În plus, am observat ce browsere sunt cele mai folosite de către populație și ce impact are acest aspect asupra folosirii diferitelor framework-uri de testare.

De asemenea, am învățat cum testarea automată prezintă multiple avantaje peste testarea manuală și ce caracteristici are fiecare tip de interfață desktop, web și mobil pentru a putea executa testarea automată în funcție de fiecare tip de interfață. Odată cu diferențele caracteristici pe care le prezintă diferențe interfețe, se poate identifica și ce fel de instrumente și framework-uri sunt indicate pentru testare în funcție de caz, precum Crawlijax pentru interfață web ce are structura de tip DOM, Gui-tar pentru interfață desktop și diferențe framework-uri pentru structura event-based specifică IOS și Android. Pe de altă parte, testarea folosind Record and playback îmbunătățește testarea automată prin înregistrarea acțiunilor pe interfața grafică și repetarea lor. Astfel, am studiat în ce constă acest mecanism și cum framework-uri precum Quick Test Professional, Selenium și Test Complete au adus îmbunătățiri în

procesul testării interfeței grafice. Un ultim tip de testare a interfeței grafice pe care l-am studiat a fost testarea folosind Robotic Process Automation (RPA), ce aduce noi caracteristici în facilitarea procesului de testare. Am studiat în ce constă testarea folosind roboti software și cum influențează productivitatea oamenilor în comparație cu alte tipuri de testare. Am descoperit ce reprezintă suita de testare UiPath și care sunt principalele componente ale acesteia, precum și cum se măsoară performanța robotilor software în îndeplinirea sarcinilor de testare. În final, am concluzionat care au fost dificultățile în testarea interfețelor grafice și cum diferă acestea comparativ, precum și care ar putea fi viitoarele direcții de dezvoltare a testării interfețelor grafice pentru combaterea dificultăților.

În altă ordine de idei, în ceea ce privește aplicarea teoriei în practică, am descoperit ce înseamnă testarea interfeței grafice a unei aplicații implementate de mine, Projy. Am învățat ce înseamnă implementarea unei aplicații pornind de la idei proprii de funcționalități până la folosirea de cele mai bune practici din industrie, cele mai noi tehnologii, framework-uri, limbaje și librării apărute ce facilitează codarea serverului și a clientului, astfel încât să se apropie cât mai mult de o aplicație ce urmează a fi lansată în producție: Spring Boot pentru server și Flutter pentru client. M-am concentrat pe implementarea unor cazuri de utilizare ce au importanță și sunt facile pentru utilizator și anume login, register, switch account, edit profile, delete account, show projects, post project, show user projects, având în vedere o diagramă a cazurilor de utilizare. Mi-am îndreptat atenția și spre partea de UI Design a aplicației astfel încât utilizatorul să aibă o experiență cât mai plăcută și interacțiunea cu interfața grafică să fie cât mai facilă și intuitivă, folosind Figma. Am pus la dispoziție un manual de utilizare în care am explicat care sunt pașii de utilizare a aplicației și o diagramă ce evidențiază arhitectura client server pentru vederea în ansamblu a aplicației. Am avut în vedere implementarea aplicației folosind baze de date relationale, atât globală cât și locală, având în considerare detalii precum conexiunea utilizatorului la internet și tratarea cazurilor de excepție. Am conceput o diagramă de clase ce ilustrează structura aplicației, împreună cu evidențierea tehnologiilor folosite în cod. De asemenea, am făcut un studiu comparativ cu ceea ce există deja în industrie, în comparație cu aplicația implementată de mine. În cele din urmă, am testat interfața grafică a aplicației folosind diferite tehnologii și am comparat tehniciile de testare folosite.

Așadar, în urma teoriei studiate din domeniul testării interfețelor grafice și în urma aplicării ei prin testarea aplicației mele, putem concluziona că testarea interfețelor grafice este foarte importantă pentru aplicații ce urmează să intre în producție și poate aduce un plus de valoare pentru utilizatorii care o folosesc deoarece, având în vedere că scopul lor este de a face viața oamenilor mai ușoară, aplicațiile lipsite de erori sunt cele care îmbunătățesc cu adevărat viața de zi cu zi a oamenilor.

Bibliografie

- [Abb11] Abbot. Abbot. Accessed: 23.04.2022, <http://abbot.sourceforge.net/doc/overview.shtml>, 2011.
- [AFT⁺12] Domenico Amalfitano, Anna Rita Fasolino, Porfirio Tramontana, Salvatore De Carmine, and Atif M Memon. Using gui ripping for automated testing of android applications. In *2012 Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, pages 258–261. IEEE, 2012.
- [AFT⁺15] Domenico Amalfitano, Anna Rita Fasolino, Porfirio Tramontana, Bryan Dzung Ta, and Atif M. Memon. Mobiguitar: Automated model-based testing of mobile apps. *IEEE Software*, 32(5):53–59, 2015.
- [Aut22] AutomationAnywhere. Automationanywhere. Accessed: 26.04.2022, <https://www.automationanywhere.com>, 2022.
- [Blu22] BluePrism. Blueprism. Accessed: 26.04.2022, <https://www.blueprism.com>, 2022.
- [CKNZ11] Edmund M Clarke, William Klieber, Miloš Nováček, and Paolo Zuliani. Model checking and the state explosion problem. In *LASER Summer School on Software Engineering*, pages 1–30. Springer, 2011.
- [CNS13] Wontae Choi, George Necula, and Koushik Sen. Guided gui testing of android apps with minimal restart and approximate learning. *Acm Sigplan Notices*, 48(10):623–640, 2013.
- [CSS20] Marina Cernat, Adelina Nicoleta Staicu, and Alin Stefanescu. Towards automated testing of rpa implementations. In *Proceedings of the 11th ACM SIGSOFT International Workshop on Automating TEST Case Design, Selection, and Evaluation*, pages 21–24, 2020.
- [CWP⁺18] Nana Chang, Linzhang Wang, Yu Pei, Subrota K. Mondal, and Xuan-dong Li. Change-based test script maintenance for android apps. In

- 2018 IEEE International Conference on Software Quality, Reliability and Security (QRS), pages 215–225, 2018.
- [Cyp22] Cypress. Cypress. Accessed: 23.04.2022, <https://www.cypress.io>, 2022.
- [Dar22] Dart. Dart. Accessed: 19.05.2022, <https://dart.dev>, 2022.
- [Dri22] Dribble. Model-based testing for ios apps. Accessed: 22.04.2022, <https://dribbble.com/shots/15476174-Model-based-testing-for-iOS-apps#>, 2022.
- [Fig22] Figma. Figma. Accessed: 19.05.2022, <https://www.figma.com>, 2022.
- [Flu22a] FlutLab. Flutlab. Accessed: 19.05.2022, <https://flutlab.io>, 2022.
- [Flu22b] Flutter. Flutter. Accessed: 19.05.2022, <https://flutter.dev>, 2022.
- [Flu22c] Flutter. Flutter architectural overview. Accessed: 23.05.2022, <https://docs.flutter.dev/resources/architectural-overview>, 2022.
- [Flu22d] Flutter. Integration testing. Accessed: 25.05.2022, <https://docs.flutter.dev/testing/integration-tests>, 2022.
- [GCZM16] Zebao Gao, Zhenyu Chen, Yunxiao Zou, and Atif M. Memon. Sitar: Gui test script repair. *IEEE Transactions on Software Engineering*, 42(2):170–186, 2016.
- [Git22] GitHub. Github. Accessed: 23.05.2022, <https://github.com>, 2022.
- [Gra22] Gradle. Gradle. Accessed: 19.05.2022, <https://gradle.org>, 2022.
- [IML09] Juha Itkonen, Mika V. Mantyla, and Casper Lassenius. How do testers do it? an exploratory study on manual testing practices. In *2009 3rd International Symposium on Empirical Software Engineering and Measurement*, pages 494–497, 2009.
- [Jav22] Java. Java. Accessed: 19.05.2022, <https://www.java.com/en/>, 2022.
- [JMV04] Natalia Juristo, Ana M. Moreno, and Sira Vegas. Reviewing 25 years of testing technique experiments. *Empirical Softw. Engg.*, 9(1–2):7–44, mar 2004.

- [Jou22] JournalDev. What is selenium? introduction to selenium. Accessed: 18.04.2022, <https://www.journaldev.com/25395/what-is-selenium-introduction-to-selenium>, 2022.
- [KFN99] Cem Kaner, Jack L. Falk, and Hung Quoc Nguyen. *Testing Computer Software, Second Edition*. John Wiley & Sons, Inc., USA, 2nd edition, 1999.
- [Khe21] Aashish Khetarpal. What is cypress: Introduction and architecture. Accessed: 23.04.2022, <https://www.toolsqa.com/cypress/what-is-cypress/>, 2021.
- [KK11] Manjit Kaur and Raj Kumari. Comparative study of automated testing tools: Testcomplete and quicktest pro. *International Journal of Computer Applications*, 24(1):1–7, 2011.
- [KM15] Emily Kowalczyk and Atif Memon. Extending manual gui testing beyond defects by building mental models of software behavior. In *2015 30th IEEE/ACM International Conference on Automated Software Engineering Workshop (ASEW)*, pages 35–41, 2015.
- [LCOLL19] Craig Le Clair, C O'Donnell, A Lipson, and D Lynch. The forrester wave™: Robotic process automation, q4 2019. *Retrieved from Forrester Research database*, 2019.
- [LCW⁺17] Xiao Li, Nana Chang, Yan Wang, Haohua Huang, Yu Pei, Linzhang Wang, and Xuandong Li. Atom: Automatic maintenance of gui test scripts for evolving mobile applications. In *2017 IEEE International Conference on Software Testing, Verification and Validation (ICST)*, pages 161–171, 2017.
- [Lin22] Linkedin. Linkedin. Accessed: 23.05.2022, <https://www.linkedin.com>, 2022.
- [Lom22] Lombok. Lombok. Accessed: 19.05.2022, <https://projectlombok.org>, 2022.
- [MA⁺19] Fatini Mobaraya, Shahid Ali, et al. Technical analysis of selenium and cypress as functional automation framework for modern web application testing. *Department of Information Technology, AGI Institute, Auckland, New Zealand*, 2019.
- [Man17] ManiacDev. Swiftmonkey. Accessed: 22.04.2022, <https://maniacdev.com/2017/01/>

- swiftmonkey-a-swift-based-framework-for-random-ui-testing, 2017.
- [Mes03] Gerard Meszaros. Agile regression testing using record & playback. In *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 353–360, 2003.
- [Mes15] Ali Mesbah. Advances in testing javascript-based web applications. In *Advances in computers*, volume 97, pages 201–235. Elsevier, 2015.
- [MP18] Inês Coimbra Morgado and Ana CR Paiva. Mobile gui testing. *Software Quality Journal*, 26(4):1553–1570, 2018.
- [MTN13] Aravind Machiry, Rohan Tahiliani, and Mayur Naik. Dynodroid: An input generation system for android apps. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2013, page 224–234, New York, NY, USA, 2013. Association for Computing Machinery.
- [MTR08] Alessandro Marchetto, Paolo Tonella, and Filippo Ricca. State-based testing of ajax web applications. In *2008 1st International Conference on Software Testing, Verification, and Validation*, pages 121–130, 2008.
- [MVDL12] Ali Mesbah, Arie Van Deursen, and Stefan Lenselink. Crawling ajax-based web applications through dynamic analysis of user interface state changes. *ACM Transactions on the Web (TWEB)*, 6(1):1–30, 2012.
- [MyS22] MySQL. Mysql. Accessed: 19.05.2022, <https://www.mysql.com>, 2022.
- [ND12] Leckraj Nagowah and Kishan Doorgah. Improving test data management in record and playback testing tools. In *2012 International Conference on Computer & Information Science (ICCIS)*, volume 2, pages 931–937. IEEE, 2012.
- [NRBM14] Bao N Nguyen, Bryan Robbins, Ishan Banerjee, and Atif Memon. Guitar: an innovative tool for automated testing of gui-driven software. *Automated software engineering*, 21(1):65–105, 2014.
- [OJPZ11] Frolin S Ocariza Jr, Karthik Pattabiraman, and Benjamin Zorn. Javascript errors in the wild: An empirical study. In *2011 IEEE 22nd International Symposium on Software Reliability Engineering*, pages 100–109. IEEE, 2011.

- [PRZ18] Mauro Pezzè, Paolo Rondena, and Daniele Zuddas. Automatic gui testing of desktop applications: An empirical assessment of the state of the art. In *Companion Proceedings for the ISSTA/ECOOP 2018 Workshops*, ISSTA '18, page 54–62, New York, NY, USA, 2018. Association for Computing Machinery.
- [Pub22a] Pub.dev. Connectivity. Accessed: 19.05.2022, https://pub.dev/packages/connectivity_plus, 2022.
- [Pub22b] Pub.dev. Moor. Accessed: 19.05.2022, https://pub.dev/packages/moor_flutter, 2022.
- [QWMW17] Xiao-Fang Qi, Zi-Yuan Wang, Jun-Qiang Mao, and Peng Wang. Automated testing of web applications using combinatorial strategies. *Journal of Computer Science and Technology*, 32(1):199–210, 2017.
- [Rep22] Repeato. Repeato. Accessed: 25.05.2022, <https://www.repeato.app>, 2022.
- [Roy20] RoyalCyber. Functional testing with robotic process automation. Accessed: 26.04.2022, <https://www.royalcyber.com/blog/devops/automation/functional-testing-with-robotic-process-automation/>, 2020.
- [RSS17] Paruchuri Ramya, Vemuri Sindhura, and P. Vidya Sagar. Testing using selenium web driver. In *2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, pages 1–7, 2017.
- [Sel22a] Selenium. Selenium. Accessed: 23.04.2022, <https://www.selenium.dev>, 2022.
- [Sel22b] SeleniumIDE. Seleniumide. Accessed: 25.05.2022, <https://www.selenium.dev/selenium-ide/>, 2022.
- [Sma21a] Smartbear. Automated ui testing that covers you from device cloud to packaged apps. Accessed: 25.04.2022, <https://smartbear.com/product/testcomplete/overview/>, 2021.
- [Sma21b] Smartbear. Testcomplete features. Accessed: 25.04.2022, <https://smartbear.com/product/testcomplete/features/>, 2021.
- [Sou14] SourceForge. Guitar - a gui testing framework. Accessed: 20.04.2022, <https://sourceforge.net/projects/guitar/>, 2014.

- [SP13] Neelam Sirohi and Anshu Parashar. Component based integration testing using abbot tool. *International Journal of Computer Applications*, 74(19), 2013.
- [Spr] [Spr22] Spring. Spring data jpa. Accessed: 19.05.2022, <https://spring.io/projects/spring-data-jpa>, 2022.
- [SqL22] SQLite. Sqlite. Accessed: 19.05.2022, <https://www.sqlite.org/index.html>, 2022.
- [Stu22] Android Studio. Monkey. Accessed: 22.04.2022, <https://developer.android.com/studio/test/other-testing-tools/monkey>, 2022.
- [Tes19] TestingWhiz. 5 reasons to use record and playback feature in test automation. Accessed: 24.04.2022, <https://www.testing-whiz.com/blog/5-reasons-to-use-record-and-playback-feature-in-test-automation>, 2019.
- [TKH11] Tommi Takala, Mika Katara, and Julian Harty. Experiences of system-level model-based gui testing of an android application. In *2011 Fourth IEEE International Conference on Software Testing, Verification and Validation*, pages 377–386. IEEE, 2011.
- [UiP20] UiPath. Rpa testing. Accessed: 26.04.2022, <https://docs.uipath.com/studio/v2020.4/docs/rpa-testing>, 2020.
- [UiP22a] UiPath. Align business outcomes with rpa operations using uipath insights. Accessed: 26.04.2022, <https://www.uipath.com/blog/product-and-updates/business-outcomes-rpa-operations-insights>, 2022.
- [UiP22b] UiPath. Test automations and applications smarter, faster, and more thoroughly. Accessed: 26.04.2022, <https://www.uipath.com/solutions/department/enterprise-test-suite>, 2022.
- [UiP22c] UiPath. Uipath. Accessed: 26.04.2022, <https://www.uipath.com>, 2022.
- [UiP22d] UiPath. Uipathstudio. Accessed: 26.04.2022, <https://www.uipath.com/product/studio>, 2022.

- [UiP22e] UiPath. Workflow design. Accessed: 26.04.2022, <https://docs.uipath.com/studio/docs/workflow-design>, 2022.
- [VKCF⁺15] Tanja EJ Vos, Peter M Kruse, Nelly Condori-Fernández, Sebastian Bauersfeld, and Joachim Wegener. Testar: Tool support for test automation at the user interface level. *International Journal of Information System Modeling and Design (IJISMD)*, 6(3):46–83, 2015.
- [Wit21] Thibault Wittemberg. A dsl for state machines in swift. Accessed: 19.04.2022, <https://twittemb.github.io/posts/2021-02-13-StateMachineDSL/>, 2021.
- [XGF08] Qing Xie, Mark Grechanik, and Chen Fu. Rest: A tool for reducing effort in script-based testing. In *2008 IEEE International Conference on Software Maintenance*, pages 468–469, 2008.
- [Yan11] Xuebing Yang. Graphic user interface modelling and testing automation. 2011.