POLITECNICO DI TORINO

Master's Degree in Ingegneria Informatica (Computer Engineering)

Master's Degree Thesis

Test Automatizzati per Software dell'Autoveicolo: analisi di un'architettura esistente e potenziali miglioramenti

Supervisor

Prof. Riccardo COPPOLA

Candidate

Federico SIVIERO

November 2022/2023

Summary

All'interno del settore automotive, il ruolo del software sta diventando sempre più preponderante. Agli autoveicoli viene richiesto di integrare sempre più funzioni, e l'interconnessione sia tra i componenti interni al veicolo sia tra il veicolo e l'ambiente circostante è sempre maggiore. Inoltre, le attuali esigenze di mercato impongono ritmi sempre più serrati per massimizzare i profitti. Tuttavia, un processo imprescindibile è quello della validazione, e la produzione di test efficaci richiede tempo e risorse. A tal proposito, si è alla ricerca di soluzioni di automazione che consentano di rilasciare il prodotto sul mercato il prima possibile. In questa trattazione vengono discussi gli aspetti fondamentali del software dell'autoveicolo, dell'automazione di test, di come questa venga applicata in un caso pratico e come questo possa essere migliorato.

L'elemento fondamentale della logica dell'autoveicolo è rappresentata dalla ECU (Electronic Control Unit). Essa ha il compito di ricevere informazioni dai sensori a cui è collegata, elaborare una risposta e inviare comandi appropriati a degli attuatori. Per svolgere funzioni più complesse, queste centraline possono comunicare tra di loro, facendo uso di infrastrutture come la rete CAN. Gruppi di ECU connesse controllano i sottosistemi fondamentali del veicolo: powertrain (motore, trasmissione), chassis (freni, sterzo, sospensioni), body (sicurezza, comfort), multimedia (radio, GPS). L'evoluzione di questa interconnessione ha portato alla creazione di veri e propri servizi connessi forniti da enti esterni, a cui il veicolo può associarsi dinamicamente.

La test automation ha lo scopo di velocizzare il processo di testing ed aumentare la sua efficacia, andando ad intervenire sulle problematiche caratteristiche del testing manuale, come ad esempio l'elevato potenziale di errore dato dal fattore umano. Per ottenere il massimo dall'automazione viene spesso creato un Test Automation Framework: una piattaforma che consente di definire librerie di funzioni, organizzare l'esecuzione di più test in parallelo e gestire la reportistica associata ai test. I framework sono classificati in diverse categorie: linear (basati su record e playback), modular (basati sulla suddivisione dell'applicazione in moduli), library architecture (basati su librerie di funzioni comuni), data-driven (focalizzati sulla gestione dei dati di test), keyword-driven (basati sulla database di parole chiave), hybrid, test-driven

(basati su Test-Driven Development), behavior-driven (basati su Behavior-Driven Development).

In base alla piattaforma a cui l'applicazione appartiene, vengono utilizzati strumenti di test automation diversi. Per il web, uno degli strumenti fondamentali è Selenium: esso si basa sull'utilizzo di un WebDriver per inviare comandi al browser ed interagire con il DOM, in modo da automatizzare gli scenari di test desiderati. Per interagire con gli elementi della pagina è fondamentale individuare del selettori adeguati: delle "query" che identificano i componenti del DOM tramite i loro attributi o la loro posizione rispetto agli altri elementi.

Uno degli strumenti fondamentali per il testing su mobile è Appium. La sua implementazione è basata sul mapping dei metodi del WebDriver di Selenium (fatta eccezione per quelle corrispondenti ad azioni prive di significato al di fuori del browser), e come esso si basa sull'utilizzo di un driver per inviare comandi all'applicazione. Inoltre, è necessario estendere le funzionalità di base a seconda della piattaforma di riferimento: nel caso di Android, ci si appoggia al sistema driver/server di UiAutomator2; nel caso di iOS, si fa ricorso alla tecnologia di XCUITest e alla sua implementazione di WebDriver (WebDriverAgent).

Reply è una società di consulenza torinese fondata nel 1996. Da allora, essa ha ampliato i suoi orizzonti e ad oggi dispone di sedi poste in tutto il mondo. La sua struttura è basata su una serie di sotto-società specializzate e, tra esse, Reply Concept si dedica a IoT e test automation.

Il processo di test automation di Reply Concept ha come suo fulcro il TAF: una piattaforma web di automazione sviluppata internamente. All'interno della piattaforma è possibile visualizzare statistiche sui test eseguiti, programmare l'avvio dei test e visualizzare i progressi di esecuzione in tempo reale. I singoli test sono implementati come funzioni Java annotate tramite Cucumber, uno strumento di BDD che consente di definire file .feature in cui ogni fase del test è rappresentata da una stringa in linguaggio naturale. Per cercare di ridurre il tempo dedicato al testing, ed eventualmente aggiungere nuove funzionalità, sono state considerate alcune soluzioni.

La prima soluzione si basa sull'introduzione di funzionalità basate sull'intelligenza artificiale. Infatti, attraverso l'uso di AI si possono ottenere risultati come il self-healing (la correzione automatica di selettori non più validi) e lo spidering (la generazione di test tramite l'esplorazione automatica dell'applicazione). A tal proposito, si è valutato l'utilizzo di uno strumento come Katalon.

Katalon è una piattaforma di test che offre una serie di strumenti di supporto alla software quality. In particolare sono compresi Katalon TestOps(pianificazione, organizzazione in test suite e reportistica), Katalon Studio (un ambiente di sviluppo per la produzione di test) e Katalon TestCloud (un ambiente di esecuzione costruito su cloud). All'interno di Katalon Studio, i test vengono principalmente generati tramite record e playback: si registrano le azioni eseguite sull'applicazione e si

genera automaticamente uno script ri-eseguibile. I test possono essere modificati sia tramite UI (nella Manual View) sia come script (nella script view). Per i test di applicazioni web è disponibile il self-healing: in caso di fallimento del selettore predefinito, vengono effettuati dei tentativi con gli altri presenti nel repository fino a trovarne uno funzionante.

Katalon consente di produrre test in modo facile e intuitivo. Tuttavia, l'assenza di self-healing per mobile risulta problematica e sarebbe necessario trovare un modo di integrare i test di Katalon nel flusso del TAF.

Un'altra alternativa, che si propone di ridurre di molto le competenze di programmazione richieste, è ECU-TEST, una piattaforma di test sviluppata da Tracetronic. Ogni progetto di ECU-TEST richiede la presenza di un Test Configuration File (in cui si definiscono i parametri delle interfacce di Appium, Selenium e altri strumenti) e un Test Bench Configuration File (che contiene impostazioni generali di ECU-TEST). I singoli test (chiamati package) sono costituiti da una sequenza di Job: azioni definite da ECU-TEST e dalle interfacce collegate. E' notevole il vantaggio in termini di competenze di programmazione richieste ma dall'automazione uno scenario distribuito tra vari dispositivi sono emersi alcuni problemi: i Job offerti dall'interfaccia di Appium operano solo su coordinate, risultando in test poco affidabili, e la verifica della presenza di elementi è eseguibile solo tramite un confronto di immagini che allunga il processo di test.

Un altro tipo di prospettiva è dato dall'integrazione di un nuovo strumento nel TAF: il Custom Feature Generator (CFG). Esso consente di creare test attraverso la costruzione di uno schema a blocchi. Ogni blocco corrisponde ad uno step Cucumber della libreria del TAF e, dopo aver disposto i blocchi necessari, un file .feature viene automaticamente generato. La creazione di questi schemi è resa possibile da flowy: un framework in linguaggio Javascript che definisce i blocchi e la logica per trascinarli nell'area di lavoro e unirli per formare una sequenza. Le funzioni di flowy di esportazione e importazione di uno schema consentono di memorizzare la sequenza di blocchi su un db e recuperarla in seguito per poterla modificare. Nell'area di lavoro, a supporto dello schema a blocchi corrente, è mostrato un riepilogo della feature che ne verrebbe creata (feature preview) per dare all'utente la possibilità di mantenere una visione di insieme e navigare tra i blocchi posizionati.

In conclusione, per ridurre il tempo necessario al testing si possono esplorare soluzioni commerciali ma esse si distaccano molto dall'architettura corrente. Non è garantito, inoltre, che queste si allineino alle esigenze di test dell'azienda. Nuove funzionalità, però, in particolare quelle legate all'intelligenza artificiale, hanno il potenziale di rivoluzionare il processo di testing attuale. Per usufruire di questi vantaggi senza stravolgere l'architettura corrente, si può tentare di includere sistemi basati su di esse all'interno del TAF corrente, ad esempio come supporto all'emergente CFG.

Acknowledgements

Ringrazio tutti coloro che mi hanno affiancato durante questo percorso, sia per le conoscenze che mi hanno impartito sia per il supporto nei momenti più diffcili.

Table of Contents

List of Figures

Acronyms

ECU
electronic control unit

ABS
automatic braking system

CAN
controller area network

ACC
adaptive cruise control

TCS
traction control system

EBD
electronic brake distribution

ESC
electronic stability control

BCM
body control module

RAM
random access memory

ROM
read-only memory

SiL

software-in-the-loop

HiL

hardware-in-the-loop

TA

test automation

AI

artificial intelligence

TAF

test automation framework

TDD

test-driven development

BDD

behavior-driven development

ADB

android debug bridge

DOM

domain object model

API

application programming interface

UI

user interface

IoT

internet of things

AoT

autonomy of things

CTO

chief techonolgy officier

BU

business unit

AV&R

autonomous vehicles & robots

MOBI

mobility open blockchain initiative

AR/VR

augmented reality/virtual reality

CFG

custom feature generator

Chapter 1

Introduzione

All'interno del settore automotive il ruolo del software è fondamentale. Esso è utilizzato non soltanto per le funzioni di bordo, ma anche per la connessione ad altri veicoli o ad enti esterni. L'avvento del software ha dato ai produttori di autoveicoli la possibilità di offrire molti dei servizi che oggi appaiono scontati, la maggior parte dei quali è associata al sistema di infotainment. La richiesta di nuove funzionalità è sempre maggiore, e le statistiche confermano che la quantità di software (in termini di numero di linee di codice) installata su una vettura è in costante aumento. Ogni produttore è in competizione con gli altri per essere il primo a portare il proprio servizio rivoluzionario sul mercato. Tuttavia lo sviluppo di software (specie nei contesti in cui la sicurezza degli utenti è a rischio) è un processo strutturato, con fasi e obiettivi ben definiti. In particolare, un ruolo essenziale è dato dalla validazione: un "male necessario" che sottrae tempo che potrebbe essere, altrimenti, dedicato allo sviluppo di nuove funzioni o alla messa sul mercato del prodotto. Vista la mole elevata di funzioni da testare, sono necessari degli strumenti per ridurre il tempo dedicato alla validazione (impattando la qualità il meno possibile) e far fronte alla complessità. Una delle soluzioni più comunemente adottate è la Test Automation: a partire da un insieme di scenari di test definiti, si realizzano dei programmi in grado di automatizzare la loro esecuzione e verificare che quest'ultima venga portata avanti correttamente. A tal proposito vengono costruiti dei Test Automation Framework, che oltre a definire librerie di funzioni di testing possono essere usati per programmare l'esecuzione dei test case, monitorare i risultati e organizzare i report associati. A tal proposito, si è sempre alla ricerca di nuovi modi per rendere il processo di testing più rapido ed efficace. Questi comprendono strumenti volti a semplificare il test design, ridurre la necessità di modificare i test case in fase di regression testing e rendere la scrittura di test in generale più semplice ed accessibile. Di seguito, si discuterà degli elementi fondamentali che compongono il software dell'autoveicolo e quali siano le fasi principali del suo processo produttivo. Quindi, si introdurrà il concetto

di Test Automation e Test Automation Framework: quali sono i vantaggi rispetto ai test manuali, quali categorie di framework esistono, quali sono alcuni degli strumenti utilizzati per l'automazione. In particolare verranno illustrati gli aspetti fondamentali di Selenium e Appium: si introdurrà il concetto di selettori (insieme alle categorie in cui si distinguono) e il modo in cui, a partire dalle specifiche definite da WebDriver, viene implementata l'automazione di browser e applicazioni per sistemi Android e iOS. Di seguito, si descriverà come i framework vengono costruiti e come le loro prestazioni vengono misurate. In seguito, verrà analizzato il caso specifico del TAF di Reply: da quali elementi è costituito, come esso gestisce l'esecuzione dei test e i suoi utenti e quali strumenti offre per la creazione di nuovi test case. Inoltre, verranno descritti alcuni casi particolari di test presenti nel TAF per mostrare alcune delle automazioni che possono essere implementate attraverso i Driver e per osservare come la piattaforma di Reply sia in grado di interfacciarsi a diverse tecnologie. Dopodichè, si esploreranno alcune delle possibili strade che possono essere percorse per migliorare il framework corrente: attraverso l'analisi della piattaforma di Katalon, si discuteranno le possibili novità introdotte dall'intelligenza artificiale (come la possibilità di ricorrere a self-healing per ottenere test script più robusti) e come la scrittura di test possa essere semplificata tramite funzioni di record-and-playback; sperimentando l'automazione di uno scenario che coinvolge diversi dispositivi tramite ECU-TEST, si discuterà come questo possa integrare diverse tecnologie e fornire un metodo di produzione di test script senza codice. Infine, si esamineranno le possibilità introdotte da un nuovo strumento in via di sviluppo presso Reply: il CFG. Verranno descritti i suoi componenti principali, come viene utilizzato e come a partire da un'interfaccia che consente la creazione di flow chart vengono creati dei test script già integrati nel TAF e pronti per essere eseguiti. Inoltre, si discuterà di possibili sviluppi futuri (sia nel breve sia nel lungo termine) in termini di nuove funzionalità e miglioramenti dell'esperienza utente. Per ognuna delle soluzioni considerate si andranno a valutare le funzionalità offerte rispetto al framework attuale, soppesando rispetto ad eventuali lacune dovute ad incompatibilità con le esigenze di test dell'azienda e alla loro praticabilità.

Chapter 2

Automotive Software

2.1 Introduzione

Negli ultimi anni, nell'industria automotive, si è osservata una crescente richiesta di nuove funzionalità con lo scopo di aumentare il livello di sicurezza e di comfort per il guidatore, oltre che un'introduzione di norme sempre più stringenti in termini di consumo di carburante, emissioni e sicurezza. A ciò si aggiunge anche il fatto che i sistemi elettronici presenti in un autoveicolo hanno bisogno di operare in contesti rigidi dal punto di vista di temperature, vibrazioni, umidità e interferenze elettromagnetiche pur garantendo un ciclo vitale relativamente lungo. La necessità di soddisfare tali richieste ha portato ad un esponenziale aumento del volume e della complessità della componente elettronica e software presente in un autoveicolo. Infatti, nel contesto automotive il settore software è quello che si è dimostrato terreno maggiormente fertile per l'innovazione. Per poter far fronte a tale complessità, si è resa necessaria l'introduzione di metodologie di sviluppo standardizzate. Un esempio notevole è lo standard ISO26262, che fornisce indicazioni sulla sicurezza dei veicoli da strada per ciascuna fase del loro ciclo produttivo. La componente elettronica di un autoveicolo è costituita da un insieme di unità dette ECU (Electronic Control Unit). Tali componenti hanno la funzione fondamentale di ricevere valori di input tramite appositi sensori, elaborare l'informazione e generare una risposta appropriata tramite una logica di controllo e trasmettere il risultato a degli attuatori che portano il veicolo ad eseguire la funzione richiesta. Infatti, ogni ECU è inserita all'interno di un sistema di controllo open/closed loop che segue il classico schema composto da sensori, attuatori, logica di controllo e impianto (il veicolo). A questo si aggiungono il guidatore e l'ambiente circostante, la cui azione modifica il comportamento del veicolo. Un esempio di ECU è, ad esempio, il sistema di controllo dei freni: a partire da informazioni come la posizione del pedale del freno, l'angolo di sterzo e la velocità di rotazione delle ruote si produce una risposta che

viene poi convertita in forza applicata dai freni sulle ruote. Inoltre, le ECU hanno la possibilità di comunicare tra loro attraverso il bus dati CAN (Controller Area Network) per scambiare informazioni e svolgere, così, funzioni complesse. Questo scambio di dati può essere anche esteso a sistemi esterni al veicolo per interagire, ad esempio, con altri veicoli e l'ambiente circostante. Tale forma di networking è stata di particolare beneficio per l'insieme dei sistemi multimediali on-board e ha portato alla nascita di funzioni come il routing dinamico per navigatori satellitari, che include informazioni sul traffico in tempo reale per stabilire il percorso più veloce.

2.2 Sistemi elettronici del veicolo

Il controllo elettronico del veicolo è affidato ad un insieme di sottosistemi, ognuno con il compito di sovrintendere una specifica sezione o macrofunzione del mezzo. Alcuni esempi sono il powertrain subsystem (che gestisce l'energia, il motore e la trasmissione), lo chassis subsystem (che coordina lo sterzo, i freni e le sospensioni), il body subsystem (che si occupa dei meccanismi di sicurezza e comfort del guidatore) e il multimedia subsystem (che include tutto ciò che concerne antenne, sistemi audio-video, sistemi di navigazione e connessione ad Internet). [1] Poco dopo la loro introduzione, le ECU erano considerate come sistemi a sé stanti: ognuno svolgeva la propria funzione in autonomia senza scambiare informazioni con altri controllori. Ogni ECU veniva assegnata ad una specifica funzione all'interno di uno specifico sottosistema del veicolo. Con l'evoluzione della tecnologia, è emersa la possibilità di sfruttare il software per offrire funzionalità più complesse e sono stati introdotti strumenti di comunicazione (come il bus CAN) che hanno facilitato la creazione di sistemi elettronici connessi. Una maggiore comunicazione ha anche consentito la condivisione di risorse come i sensori e loro segnali, riducendo quindi la quantità di hardware necessaria e i costi ad essa associati. Tuttavia, la condivisione ha anche introdotto della concorrenza tra ECU per l'accesso agli attuatori Nell'ambito dei sistemi connessi, l'implementazione delle funzioni software prevede due distinti approcci[2]:

1. La funzione è confinata in un singolo sottosistema e la sua esecuzione è delegata ad un insieme di ECU appartenenti a quel sottosistema

2. La funzione coinvolge molteplici sottosistemi ed è eseguita da ECU appartenenti a sottosistemi diversi. Molti sistemi di supporto alla guida come l'Adaptive Cruise Control rientrano in questa categoria. In questo caso, sono necessari particolari unità dette gateway ECU che gestiscono la comunicazione tra sottosistemi

2.2.1 Sistemi del powertrain

I sistemi del powertrain rappresentano un progresso fondamentale nel controllo del veicolo. Il sistema di trasmissione, ad esempio, deve assicurarsi che il momento fornito dal motore venga trasferito alla strada in modo da ottenere il livello di trazione richiesto dal guidatore. Il livello di controllo ottenuto tramite sistemi elettronici ha portato all'introduzione di tecnologie come il cambio automatico. Inoltre, il controllo elettronico della trasmissione risulta più efficiente in termini di consumo di carburante. Il controllo elettronico è anche fondamentale, ad esempio, nei veicoli ibridi per la coordinazione tra il motore a combustione e quello elettrico I sistemi elettronici del powertrain sono controllati da ECU dedicate al motore e alla trasmissione, che si basano sia su input forniti dall'utente sia su segnali provenienti da sensori per controllare gli attuatori di accensione, iniezione, valvola della frizione e altri sistemi. Le informazioni fornite dal guidatore comprendono la posizione del pedale dell'acceleratore, quella del pedale della frizione e la marcia selezionata dalla leva del cambio, mentre dei sensori captano informazioni come pressione, temperatura, velocità, livello di tensione della batteria. La comunicazione tra tutte queste entità è gestita dalla rete CAN. In generale, l'intero sottosistema è affidato ad un ristretto numero di ECU in grado di eseguire una grande quantità di funzioni software. Tra queste, l'unità fondamentale è la ECU del motore. La ECU del motore è la più grande e complessa. Essa racchiude una grande quantità di funzioni software, che devono non solo coordinarsi tra loro ma anche interfacciarsi con funzioni del body subsystem o chassis subsystem. Il livello di prestazioni richiesto dal software è molto elevato: si richiede di eseguire funzioni con numerosi parametri che possono includere mappe o curve caratteristiche in tempi brevi. Alcuni degli attuatori collegati a questa ECU sono le candele, il controllo della frizione, gli iniettori di carburante, la ventilazione del serbatoio e altre valvole di aerazione.

2.2.2 Sistemi dello chassis

I sistemi dello chassis includono, fondamentalmente, i sistemi dei freni, dello sterzo e di controllo delle sospensioni e hanno grande influenza sulla stabilità del veicolo. L'obiettivo fondamentale di questi sistemi è assicurare il comfort e la sicurezza del guidatore. Infatti, i requisiti di sicurezza per questi sistemi sono molto stretti e il loro design è guidato da standard rigidi. Le ECU dei freni gestiscono il controllo di sistemi come Automatic Braking System (ABS), Traction Control System (TCS), Electronic Brake Distribution (EBD). Tali sistemi vengono controllati sia individualmente sia in coordinazione tra loro e ECU di motore, trasmissione e ruote. La complessità delle ECU è, in generale, minore rispetto a quella del motore. Una ECU per l'ABS, ad esempio, acquisisce informazioni come la tensione della batteria, la velocità delle ruote (attraverso sensori posti su ciascuna ruota) e lo stato dell'interruttore della luce dei freni per applicare un'azione frenante su ogni

singola ruota agendo sulle valvole solenoidali che regolano la pressione dell'olio dei freni. Un ulteriore esempio è il sistema di Electronic Stability Control (ESC), che monitora l'accelerazione longitudinale e laterale del veicolo, la velocità di rotazione delle ruote e l'angolo di sterzo per intervenire sui freni di ciascuna ruota o sulle sospensioni e migliorare la stabilità. ECU dello chassis più avanzate possono anche includere funzionalità come Predictive Emergency Braking System, che è in grado di informare il guidatore in caso di rischio di collisione e, se necessario, applicare la massima forza frenante possibile (ovvero la massima forza applicabile senza bloccare le ruote) per arrestare il veicolo.

2.2.3 Sistemi del body

I sistemi del body si suddividono in due categorie fondamentali: i sistemi dedicati al comfort/benessere degli occupanti e quelli di cosiddetta sicurezza passiva. I sistemi del body subsystem includono, ad esempio, le ECU che controllano porte, finestrini, specchietti, sedili, luci, ventilazione, riscaldamento, aria condizionata, cruise control e spie del cruscotto. Possono anche essere incluse funzionalità più avanzate come memoria della posizione dello sterzo e riscaldamento dei sedili. I sistemi di sicurezza passiva includono tutti i dispositivi onboard che si occupano di ridurre il livello di rischio dei passeggeri in caso di incidente. A questo gruppo appartengono, ad esempio, ECU degli airbag con sistema di rilevazione occupanti, regolatori di tensione delle cinture di sicurezza e roll-bar attivi. Per quanto riguarda l'interazione con il guidatore, i sistemi di comfort funzionano principalmente sulla base di input forniti dall'utente (a cui espongono interruttori, bottoni, manopole, ecc.) mentre i sistemi di sicurezza passiva lavorano quasi esclusivamente con informazione proveniente da sensori posti sul veicolo, in modo trasparente agli occupanti. All'interno del body subsystem esiste un'elevata quantità di funzioni software che opera in modo indipendente, Tali funzioni sono di complessità ridotta e sono eseguite da ECU più semplici che possono quindi essere aggiunte o rimosse in modo modulare (riflettendo la grande quantità di meccanismi di comfort che possono essere inclusi nel veicolo, i quali richiedono una potenza computazionale decisamente inferiore rispetto al controllo del motore o della trasmissione). Infatti, questo sottosistema solitamente include una grande quantità di ECU. Tali unità vengono posizionate in diversi punti del veicolo, vicine ai sensori ed attuatori con cui devono interagire. Vista la loro quantità, non è infrequente incorrere in problemi di spazio (considerando anche che quest'ultimo varia di molto da un veicolo all'altro). Non sono, comunque, escluse funzionalità basata sul coordinamento di varie ECU, come ad esempio il sistema di chiusura centralizzata. Tale coordinazione è gestita dal Body Control Module (BCM), un nodo a cui sono connessi i sistemi del body e che funge anche da ponte di comunicazione tra le diverse reti che possono essere presenti nel veicolo (es. Ethernet e CAN).

2.2.4 Sistemi multimedia

I sistemi multimedia includono tutte le funzionalità offerte da infotainment onboard come radio, antenne, lettori CD, sistemi audio/video, GPS, sistemi di navigazione, integrazione con smartphone. Per questi sistemi sono necessarie sia una buona comunicazione con gli altri sistemi elettronici del veicolo (in modo da fornire all'utente, ad esempio, un controllo più semplice e intuitivo dei sistemi di comfort) sia verso le reti esterne (per accedere a servizi di telecomunicazione). Questo punto di accesso verso l'esterno rappresenta, ovviamente, una possibile vulnerabilità che va protetta per evitare attacchi a sistemi critici come chiusura delle porte, accensione, motore e freni. Il settore multimedia ha vissuto e sta vivendo una notevole espansione nell'ultimo periodo: il mercato richiede l'introduzione di sempre più funzionalità, il che ha avuto un effetto negativo sui tempi di sviluppo. Inoltre, con il passare del tempo sempre più funzioni del veicolo vengono integrate nel sistema di infotainment in modo da fornire agli utenti un'interfaccia maggiormente intuitiva in forma di touch display.

Esempio sistema ACC

Il sistema di Adaptive Cruise Control rappresenta un'evoluzione del Cruise Control di base, nonché un esempio delle funzionalità che possono essere ottenute attraverso la comunicazione di varie ECU. ACC sfrutta un radar frontale per identificare distanza e velocità relativa del veicolo di fronte. Quindi, la ECU calcola l'accelerazione da impartire al veicolo per mantenere una distanza di sicurezza e interviene sulla ECU del motore per influenzare il momento generato, sulla ECU della trasmissione e su quella dei freni. Si tratta, quindi, di una funzione che coinvolge i sottosistemi di powertrain e chassis.

2.2.5 Tendenze recenti

Negli ultimi anni, grazie ad un incremento delle capacità computazionali e della connettività dei microcontrollori, il numero medio di ECU contenute in un veicolo è aumentato, così come anche il numero di funzioni implementate da ciascuna di esse. Infatti, come menzionato in precedenza, agli autoveicoli viene richiesto di offrire sempre più funzionalità e molte delle funzioni svolte da sistemi prettamente idraulici o meccanici stanno gradualmente venendo affidate al software. Tuttavia, ciò comporterebbe non solo un aumento dei costi (dato dalla necessità di aggiungere nuove ECU) ma anche un peggioramento del problema dello spazio di installazione. Per questi motivi, è ragionevole aspettarsi un accentrarsi delle funzioni svolte da tante ECU semplici in una singola unità, continuando quindi la tendenza positiva del numero di funzioni svolta da una singola ECU.

2.2.6 Integrazione tra componenti e servizi connessi

Con il passare del tempo gli autoveicoli diventano sistemi sempre più integrati e interconnessi, al punto che essi possono essere considerati come dei portatori di servizi da richiedere o offrire ad altri veicoli o all'ambiente circostante (ad esempio, la condivisione di informazioni sul traffico). Per poter gestire questi servizi i componenti del veicolo devono essere coordinati tra loro, ed è necessario considerare una situazione in cui questi possono essere aggiunti, rimossi o modificati nel tempo. Ciò si traduce nella necessità di esplorare e negoziare i servizi disponibili, prima di potersi associare ad essi. Si identificano due approcci fondamentali nella gestione dei componenti connessi: *publish-subscribe* e *service-oriented*. L'approccio publish-subscribe viene utilizzato principalmente per la connessione intra-veicolo. Ogni componente invia segnali (eventi) ad un nodo centrale detto dispatcher, che si occupa di inoltrare gli eventi generati a tutti i componenti che sono in ascolto (registrati) per quei determinati segnali. Con questo approccio, se l'insieme di eventi da gestire rimane invariato, è possibile modificare liberamente l'insieme di componenti on-board. Ciò si allinea perfettamente con il formato dell'offerta dei servizi a bordo del veicolo, basata su un catalogo da cui è possibile scegliere liberamente. Il modello service-oriented è particolarmente indicato per gli scenari di comunicazione veicolo-veicolo o veicolo-ambiente. In questo caso l'interazione è basata sulla scoperta e la negoziazione di servizi con componenti esterni. Si possono individuare tre elementi fondamentali[2]:

1. **Service providers:** elementi che interagiscono con dei client e offrono delle funzioni software. Questi possono essere rappresentati, ad esempio, da dei servizi Web;

2. **Service Requestor:** elementi che richiedono servizi ai provider;

3. **Discovery agencies:** elementi intermedi tra fornitori e utilizzatori dei servizi, hanno il compito di raccogliere le informazioni sulle funzioni offerte dai provider, classificarle e renderle disponibili ai client. Questi ultimi forniscono l'insieme dei parametri del servizio a cui desiderano connettersi e la discovery agency restituisce le informazioni necessarie a stabilire la comunicazione tra client e provider.

La risposta della discovery agency può anche essere associata a politiche di qualità del servizio, come ad esempio costo e affidabilità. Questo tipo di approccio all'offerta di servizi consente di soddisfare richieste in modo dinamico: ad esempio, un veicolo che necessità di un servizio di navigazione per trovare la strada più veloce verso la destinazione può associare alla sua richiesta informazioni riguardo alla propria posizione corrente. In questo modo, l'itinerario può essere stabilito

anche in base alle condizioni di traffico in tempo reale nella zona in cui si trova il veicolo richiedente.

2.3 Sviluppo software per sistemi elettronici

Lo sviluppo di sistemi elettronici per autoveicoli si basa sulla loro suddivisione in sottosistemi (ad esempio ECU hw/sw, sensori e attuatori), che vengono sviluppati separatamente, e la loro graduale integrazione nel sistema complessivo previa validazione di ciascun componente. All'interno di questo ambito, sempre più importanza viene data al compito svolto dal software. Infatti, affidare ad esso gli aspetti funzionali del sistema significa disporre della massima libertà di implementazione di funzioni di controllo e non doversi curare di problemi come spazio (in senso fisico) di installazione. Ciò non significa che il software sia del tutto indipendente dalle problematiche tipiche dell'hardware: ad esempio, la continua spinta verso il raggiungimento dei massimi volumi di produzione al minimo dei costi porta ad una spesa sempre minore per le risorse hw, costringendo il software a raggiungere elevati livelli di ottimizzazione per non dover rinunciare a delle funzionalità. Un altro vantaggio consiste nella possibilità di apportare modifiche al software (e, quindi, alle funzionalità) attraverso aggiornamenti che possono essere installati attraverso i sistemi di diagnostica off-board e propagati internamente a ciascuna ECU grazie al collegamento esistente tra loro. Ciò risulta decisamente più rapido ed economico della sostituzione fisica delle ECU. Una preoccupazioni principali nello sviluppo di funzionalità nel campo dell'automotive è la sicurezza: è di fondamentale importanza che il livello di rischio per gli occupanti del veicolo venga mantenuto entro livelli accettabili in ogni momento. A tal proposito esistono numerosi standard (come ISO 26262) che definiscono requisiti fondamentali per qualsiasi sistema che viene integrato in un autoveicolo. Per questo motivo, è necessario che ogni parte del veicolo sia il risultato di un processo di sviluppo strutturato e comprensivo di ogni fase. A tale scopo, uno dei processi più comunemente utilizzati è il V-Model. Infatti questo modello, originariamente concepito per i sistemi embedded, attribuisce la stessa importanza a hardware e software e prevede testing sia di ciascuna delle due parti sia della loro successiva integrazione. Il processo può essere suddiviso in due modelli fondamentali[1]:

1. Un livello hardware che prevede da un lato l'analisi dei requisiti e la definizione dell'architettura del sistema e dall'altro l'integrazione di hardware software con relativo testing

2. Un livello software che prevede da un lato analisi dei requisiti software, design dei componenti software e loro implementazione e dall'altro testing dei singoli componenti software e della loro integrazione

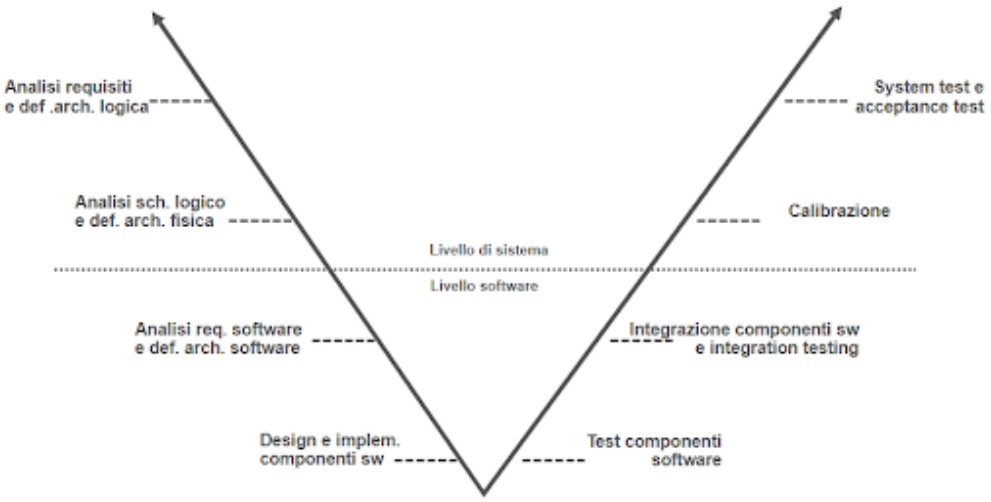

Figure 2.1: Schema del processo di sviluppo

Di seguito sono presentate le fasi fondamentali del processo.

2.3.1 Analisi dei requisiti e definizione architettura logica

A partire dai requisiti definiti secondo il punto di vista dell'utente finale, si definiscono le funzioni che devono essere offerte dal sistema e come queste interagiscono tra loro (senza considerare dettagli implementativi). I requisiti del sistema possono essere suddivisi in requisiti funzionali e requisiti non funzionali. I requisiti funzionali includono tutte le funzioni standard (eseguite quando il sistema è in condizioni normali) e non standard (eseguite in risposta a situazioni anomale come malfunzionamenti). I requisiti non funzionali, invece, coprono tutto ciò che esula dalla funzione principale offerta dal sistema: includono obiettivi di prestazioni, efficienza e scalabilità, requisiti legali derivanti da legislazioni (principalmente in materia di sicurezza) e vincoli derivanti dal contesto in cui opera il sistema (spazio di installazione, livello di tensione disponibile a bordo del veicolo). L'insieme di tutti i requisiti viene convertito in una rappresentazione schematica (il più chiara e completa possibile) delle funzioni ad alto livello eseguite dal sistema e di come queste interagiscono tra loro.

2.3.2 Analisi dello schema logico e definizione dell'architettura fisica

A partire dalla rete di funzioni definita nel passo precedente, si analizzano diverse possibilità di implementazione e si assegnano le funzioni definite in precedenza a

sottosistemi e componenti fisici. Questa operazione viene eseguita per ogni livello di astrazione del sistema, ed è infatti a questo punto che vengono definiti i requisiti specifici per hardware e software. In questa fase devono anche essere presi in considerazione requisiti di natura tecnica e economica. Ad esempio, per favorire il riutilizzo di sistemi all'interno di una specifica serie di veicoli (e, quindi, ridurre i costi) si può decidere di adoperare ECU per motore e trasmissione standardizzate. Oppure, per poter presentare una distinzione tra servizi base e optional, è necessario necessario che alcune funzioni vengano implementate in ECU separate, mentre ciò che compone il pacchetto base può essere raggruppato in una singola ECU.

2.3.3 Analisi dei requisiti software e definizione dell'architettura software

Un processo simile a quello compiuto in precedenza viene eseguito per il software. Vengono quindi definiti lo scope, i componenti software e le loro interfacce, stratificazioni del software e modi operativi. Identificare i limiti del software è fondamentale per poterne definire le interfacce di input e output verso gli altri sistemi (ad esempio, le interfacce di ciascuna ECU verso sensori, attuatori, e altri sistemi elettronici a bordo e le interfacce verso entità off board come strumenti di debugging o di diagnostica). Definire dei software layer consente di predisporre un flusso di comunicazione definito tra i vari componenti software: elementi che appartengono allo stesso livello interagiscono di frequente tra loro; si può favorire un flusso di informazione top-down, con interazione possibilmente ristretta ai livelli adiacenti (come nel modello ISO/OSI, che usato come riferimento per il modello AUTOSAR). I componenti software devono considerare due flussi di informazione distinti: il data flow, e il control flow. Il data flow descrive come l'informazione viene scambiata tra i componenti software e come questa viene processata. Processare i dati richiede la definizione di un apposito data model. Le strutture dati tipicamente utilizzate dalle applicazioni su veicolo sono valori scalari, vettori monodimensionali e matrici bidimensionali. Queste possono essere utilizzate per rappresentare strutture più complesse come curve caratteristiche o mappe caratteristiche, ad esempio evidenziando la relazione tra le variabili di input e quelle di output. Il control flow regola l'esecuzione delle istruzioni: per ogni componente software è necessario definire la sequenza di esecuzione, branch e iterazioni presenti nel flusso di esecuzione e chiamate a funzioni definite da altri componenti. La necessità di definire stati operativi del software nasce principalmente per ragioni di sicurezza: oltre al comportamento del sistema in condizioni ordinarie, in cui le funzioni di controllo vengono eseguite normalmente, è necessario prevedere delle situazioni operative in cui la normale esecuzione viene bloccata. Oltre ai singoli stati operativi, è necessario definire quali siano le transizioni possibili tra uno stato e l'altro. Il tutto, quindi, porta gli stati operativi ad essere rappresentati

tramite una macchina a stati. Ad esempio, considerando il software che controlla il quadro strumenti del veicolo, alcune delle situazioni in cui avviene un cambiamento dello stato operativo sono: l'aggiornamento software, l'esecuzione di funzioni di diagnostica e lo spegnimento del motore. Un altro aspetto da considerare è il comportamento real-time: ogni istruzione deve essere assegnata ad un processo, il quale a sua volta si unisce ad altri processi per costituire un task: un flusso di esecuzione attivato in un determinato istante di tempo.

2.3.4 Design ed implementazione dei componenti software

Durante questa fase, tutti i dettagli implementativi di ciascun componente software vengono definiti e concretizzati. Una caratteristica tipica del software per ECU è la differenziazione tra versione del software e versione dei dati. La versione dei dati fa riferimento a tutta l'informazione che rimane invariata durante l'esecuzione, come i parametri utilizzati dalle funzioni di controllo, ed essa ha spesso la necessità di essere modificata in momenti diversi rispetto al software, ad esempio durante la calibrazione delle funzioni software di uno specifico veicolo. Inoltre, questa separazione consente di adattare lo stesso software a diversi veicoli mantenendo le funzioni costanti e modificando i soli dati. Ciò porta dei benefici in termini sia di costi sia di tempo, in quanto è sufficiente che la versione software venga validata una sola volta per tutti i veicoli. Dal punto di vista dell'implementazione del data model, questa separazione si ritrova nella distinzione tra le variabili (i dati del programma) che vengono lette e scritte sulla memoria RAM e i parametri statici memorizzati nella ROM. Come menzionato in precedenza, il software per ECU deve spesso fare i conti con risorse hardware limitate per questioni economiche. Ciò si traduce, all'atto pratico, in vincoli su quantità di RAM e ROM utilizzate e tempo di esecuzione. Il requisito sulla RAM richiesta è tenuto in particolare considerazione poiché il costo della memoria è molto elevato. Per far fronte a queste restrizioni, vengono adottate numerose tecniche di ottimizzazione del software. Un esempio è la scelta di distribuzione delle sezioni del programma sulle diverse memorie presenti: sezioni di codice richiamate più di frequente vengono assegnate a memorie a tempo di accesso minore, come la ROM, mentre procedure eseguite più di rado vengono collocate su memorie più lente come la memoria Flash. Un'altra forma di ottimizzazione consiste nella separazione tra funzioni on-board e off-board: tutto il processing che produce risultati non necessari al funzionamento della rete di ECU, bensì utili soltanto in fase di diagnostica o calibrazione, vengono affidati ai tool esterni che eseguono questo tipo di operazioni.

2.3.5 Testing dei componenti software

Durante questa fase si eseguono test sui singoli componenti software sviluppati in precedenza. I metodi di test utilizzati comprendono sia quelli di tipo statico (come peer-review, analisi statica, analisi di data flow e control flow) sia quelli di tipo dinamico (come lo stress testing).

2.3.6 Integrazione di componenti software e integration testing

Ora che i singoli componenti software sono stati implementati e validati, questi possono essere gradualmente integrati a formare un sistema software che verrà in seguito testato. L'approccio adoperato per i test può seguire vari criteri: si possono definire test sulla base delle diverse funzioni software (ad esempio stabilendo una distinzione tra test di funzioni di controllo e test di funzioni di monitoring e diagnostica); sulla base dei diversi componenti presenti nel sistema (come il sottosistema di networking e il sottosistema di diagnostica) o sulla base della situazione operativa (distinguendo tra casi nominali, casi limite e situazioni di failure). Durante questa fase, vengono utilizzati strumenti come simulatori, test bench e prototipi di veicolo o veicoli di produzione. Oltre ai processi di validazione impiegati in questo momento dello sviluppo, ne possono essere impiegati altri come la creazione di un prototipo durante la fase di design dei componenti software al fine di ottenere un feedback sulla validità del design prima di procedere all'implementazione vera e propria. L'obiettivo generale è quella di passare gradualmente da un sistema in cui i singoli componenti e l'ambiente esistono come modelli eseguiti da un simulatore a uno in cui ogni modello è stato sostituito dalla corrispondente implementazione: inizialmente, si hanno dei modelli delle funzioni di controllo che interagiscono con un modello del sistema da controllare all'interno di un simulatore (in totale assenza di requisiti real-time come il tempo di esecuzione della funzione di controllo). Andando ad integrare l'implementazione dei componenti software nel sistema, si ottiene uno schema definito Software-in-the-Loop (SiL). Sebbene l'ambiente sia ancora presente in forma di modello virtuale e non vi siano ancora requisiti real-time per la simulazione, è importante componenti come i modelli delle ECU abbiano un comportamento più fedele possibile a quello reale, in modo da validare in modo efficace l'implementazione delle loro funzioni software. Infatti, SiL consente di effettuare test su di esse (eventualmente insieme ad analisi di coverage) prima della loro integrazione nell'hardware della ECU. Una volta che hardware e software della ECU sono stati validati e la loro integrazione è stata completata, è possibile passare a scenari di test in parte reali e in parte virtuali. Questo approccio prende il nome di Hardware-in-the-Loop (HiL). Nel dettaglio, il funzionamento di una ECU reale viene testato rispetto ad una simulazione del veicolo in cui deve essere

integrata. Rispetto alla fase precedente, le funzioni di controllo e le capacità di comunicazione on-board e off-board vengono testate nei loro aspetti real-time. L'approccio HiL prevede diversi possibili scenari: si può testare una singola ECU, per concentrarsi sul comportamento delle funzioni software e sul processing dei segnali di input (simulati sia per condizioni di guida ordinarie sia per casi limite); si può testare una ECU quando connessa ai suoi reali sensori ed attuatori, in modo da testare le funzionalità di controllo della ECU nella loro interezza verificare i segnali presenti su sensori e attuatori (grazie all'utilizzo di sensori aggiuntivi) e i risultati intermedi prodotti dal software di controllo; si può testare un insieme di ECU connesse tra di loro, per validare funzionalità distribuite e comunicazione attraverso il bus (ECU non ancora implementate possono incluse in forma di modelli virtuali all'interno del simulatore). Il passo successivo dell'integrazione consiste nell'utilizzo delle cosiddette test bench: per testare il comportamento real time rispetto ad uno specifico elemento del veicolo (ad esempio il motore), questo aggiunto all'insieme dei "componenti reali" (il resto del veicolo resta in forma di modello su simulatore). Proseguendo l'integrazione si passa da test bench all'impiego di prototipi e veicoli sperimentali, avendo quindi ogni componente (compreso il guidatore e l'ambiente) presente nella sua controparte reale. Durante questa fase, oltre a sistemi di misurazione e diagnostica per l'acquisizione dati vengono impiegati strumenti di calibrazione, al fine di perfezionare i parametri di controllo di ciascuna ECU.

2.3.7 Calibrazione

Durante questa fase si esegue l'ottimizzazione dei parametri interni di ciascuna ECU allo scopo di generare la data version (l'insieme di dati statici usati dal software di controllo per eseguire le proprie operazioni). Questi parametri prendono la forma di valori scalari, mappe o curve caratteristiche possono assumere valori diversi per ciascuna delle molteplici varianti di un veicolo, per cui questa operazione deve essere ripetuta per ciascuna di esse. (spesso con l'ausilio di appositi strumenti di calibrazione). Il fatto che tali varianti siano così numerose è dovuto ad una serie di fattori: principalmente, si tratta di una questione di personalizzazione (si vuole dare all'acquirente di un veicolo la possibilità di scegliere tra diverse opzioni al fine di soddisfare al meglio le sue esigenze). Inoltre, la necessità di varianti emerge dall'esistenza di regolamentazioni ed esigenze di mercato diverse per ogni nazione. I processi di calibrazione si distinguono tra offline ed online. Nel caso della calibrazione offline, l'esecuzione del software della ECU viene interrotta ogni volta che i parametri interni vengono modificati. Ciò riduce il rischio che situazioni inattese emergano durante il test, ma risulta anche sconveniente in scenari che prevedono l'utilizzo di test bench o dello stesso veicolo in quanto è necessario interrompere e riprendere le operazioni ad ogni step di calibrazione. Nel processo

online, invece, vi è la possibilità di modificare i parametri interni dinamicamente senza la necessità di interrompere le operazioni di test. Ciò risulta, ovviamente, in una procedura di calibrazione più fluida ma nella possibilità che la variazione dei parametri produca eccezioni nel software. A quest'ultimo, quindi, è richiesto un elevato livello di robustezza. Inoltre, nel caso di sistemi con importanti requisiti di sicurezza (come l'ABS), la modifica online dei parametri viene effettuata solo quando tali sistemi non sono in funzione. Per consentire una tale dinamicità di variazione dei parametri sono necessari dei supporti ulteriori, come ad esempio la creazione di un segmento di RAM condiviso tra il microcontrollore e lo strumento di calibrazione (detta calibration RAM o CAL-RAM).

2.3.8 System test e acceptance test

Infine, vengono eseguiti dei test di sistema. Il system test verifica la compatibilità tra il sistema completo e l'architettura logica del sistema, mentre i test di accettazione verificano la consistenza con i requisiti utente. Questi test possono consistere in prove su strada, eseguite nel maggior numero possibile di condizioni ambientali (giorno, notte, terreno asciutto, terreno bagnato, aree urbane, autostrade, campagna, ...).

Chapter 3

Test Automation

3.1 Introduzione

3.1.1 Perchè la spinta verso l'automazione?

L'obiettivo fondamentale del software testing è assicurarsi che un dato prodotto software corrisponda ai requisiti posti su di esso, identificando i difetti presenti tramite confronto tra i risultati generati dal software e quelli attesi. Questo processo comprende il design, l'implementazione e l'esecuzione di test ed è un aspetto fondamentale di qualsiasi progetto software. In particolare il processo di testing, così come lo sviluppo software, è descritto in termini di scope, tempo e costo[3]:

1. Lo scope definisce la porzione di software che si decide testare. Ridurlo comporta un risparmio di tempo, ma anche il fatto che alcune funzionalità non vengano validate

2. Avere maggior tempo a disposizione consente di testare di più e ottenere una qualità superiore, ma comporta un aumento dei costi

3. Sia lo scope sia il tempo influiscono direttamente sui costi. Voler ridurre la spesa implica una riduzione di tempo e/o scope e, conseguentemente, un calo della qualità

Nello scenario odierno, vi è una sempre crescente tendenza alla riduzione dei tempi di sviluppo (ogni prodotto deve arrivare sul mercato il prima possibile) è ciò ovviamente contrasta con l'obiettivo di raggiungere un livello di qualità adeguato. Per ovviare a questo problema, si può ricorrere a strumenti di test automation. Il test automation (TA) si basa sull'utilizzo di software, separato rispetto a quello da testare, per automatizzare una procedura di testing manuale o la sequenza di

azioni eseguite da un utente durante l'utilizzo di un'applicazione. Esso può essere impiegato in diverse delle fasi del processo generale di testing, come ad esempio generazione di test, functional testing, regression testing o verifica dei risultati di test. Il test automation è, quindi, ampiamente utilizzato in moltissimi processi di sviluppo software per ridurre i costi a lungo termine, migliorare l'efficienza e ridurre il tempo speso in regression testing.

3.1.2 Confronto con manual testing

Il manual testing, come suggerisce il nome, consiste nella validazione manuale del software. Ciò avviene, solitamente, per mezzo di una persona che esplora l'applicazione dal punto di vista dell'utente finale e cerca di coprire sia gli scenari ordinari sia i casi limite. Alcuni dei passaggi fondamentali del manual testing sono analisi dei requisiti, valutazione del rischio, identificazione di scenari, sviluppo dei test case, esecuzione dei test, verifica dei risultati ottenuti e compilazione dei difetti trovati. Ognuna di queste operazioni deve essere eseguita da una persona fisica. Questi sono alcuni dei vantaggi del manual testing:

1. E' efficace nel caso di testing esplorativi poiché essi richiedono un elevato livello di conoscenza del dominio e creatività
2. Rappresenta un'opzione valida nel caso di progetti a breve termine e che non prevedono un supporto prolungato nel tempo, poiché in casi come quelli il risparmio di tempo di testing introdotto dall'adozione di test automation non sarebbe sufficiente a compensare l'investimento iniziale
3. E' efficace per testing di User Experience e usability poiché sono innatamente soggettivi e necessitato di una presenza umana
4. Non introduce alcuna dipendenza su strumenti o infrastrutture esterne

Per contro, alcuni evidenti svantaggi sono:

1. E' un processo che può richiedere un grande investimento di tempo se adottato in progetti di sviluppo di scala medio-grande
2. Si tratta di una procedura potenzialmente ripetitiva e tediosa (un aspetto importante vista la necessaria presenza umana)
3. E' richiesto un elevato investimento in termini di risorse umane (i tester devono essere assunti ed adeguatamente formati)
4. L'accuratezza può essere scarsa vista l'elevata probabilità di commettere errori (sempre a causa della presenza umana)

I test automatizzati, come menzionato in precedenza, si basano invece sull'utilizzo di strumenti software specifici. Ciò porta immediatamente ad una serie di vantaggi:

1. Il testing diventa molto più veloce ed affidabile grazie alla rimozione del fattore umano (maggiore causa di errore)

2. L'automazione rende i test ripetibili e riutilizzabili (gli stessi test possono essere eseguiti più e più volte con costo aggiuntivo trascurabile e in modo consistente) e facilmente portabili su diverse piattaforme

3. Si possono eseguire molteplici test in parallelo, riducendo il tempo necessario per la loro esecuzione e consentendo, quindi, ad un'applicazione di essere rilasciata più in fretta

4. Si possono generare automaticamente report di esecuzione completi e dettagliati, con statistiche accurate in merito a percentuale di test superati/falliti e coverage.

5. L'elevata consistenza rende possibili procedure come performance testing

6. E' particolarmente indicato per progetti di larga scala, in cui il tempo risparmiato per fasi come esecuzione dei test e regression testing compensa ampiamente l'investimento iniziale

Tuttavia, occorre anche evidenziare alcuni svantaggi come:

1. L'elevato costo degli strumenti di test automation data la necessità di stabilire un'infrastruttura e mantenerla nel tempo

2. La stabilità richiesta all'applicazione da testare, necessaria per poter eseguire i test automatici in modo consistente

3. Il rischio di introdurre nuovi difetti a causa dell'adozione di strumenti software esterni all'applicazione

Ciò che emerge da questo confronto, quindi, è che test automation ha la possibilità di migliorare i processi di test sotto i punti di vista di tempo, affidabilità e consistenza ma al netto di elevati costi iniziali e di mantenimento che risultano poco sostenibili per progetti di piccola scala. Continuano ad esistere, inoltre, procedure di test meno formali che poco si prestano ad essere automatizzate.

3.2 Test automation frameworks

Nel caso in cui si decida di adottare test automation nel proprio progetto software, per ottenere il massimo dei risultati è raccomandata la costruzione di un Test Automation Framework(TAF). Esso consiste in una piattaforma che integra vari componenti hardware e software e consente, ad esempio, di: definire librerie di funzioni che possono essere utilizzate da ogni test case, avviare molteplici test in parallelo e monitorare il loro stato di esecuzione, mantenere uno storico di report dettagliati per ogni esecuzione di test. In generale, un TAF consente di sviluppare test automatizzati in modo più efficiente e strutturato, favorendo un grande riutilizzo di codice e componenti software e consentendo un'esecuzione più rapida. I test automation framework si suddividono in numerose categorie, tra cui le più ampiamente utilizzate sono[3]:

1. Linear
2. Modular
3. Library Architecture
4. Data-driven
5. Keyword-driven
6. Hybrid
7. Test-driven development
8. Behavior-driven development

3.2.1 Linear testing framework

Il framework "lineare" si basa principalmente sull'utilizzo di strumenti di record e playback. Essi sono in grado di registrare le azioni eseguite da un utente su un'applicazione e le risposte da essa fornite e generare uno script che può riprodurre l'intera sequenza di interazione ogni volta che viene eseguito. Vengono poi generati dei report sulla base dell'esecuzione di questi script. Si tratta di un framework molto semplice, che non richiede la scrittura di codice o la presenza di conoscenze tecniche avanzate e può generare dei test molto rapidamente. Per contro, i linear framework risultano poco scalabili, in quanto richiedono un'elevata attività manuale e presentano elevati costi di mantenimento (se l'applicazione da testare subisce cambiamenti importanti, tutti gli script vengono invalidati o, perlomeno, necessitano di essere aggiornati).

3.2.2 Modular testing framework

I framework modulari si basano sull'analisi dell'applicazione da testare al fine di identificare dei moduli, da raggruppare secondo diversi livelli di astrazione, e sviluppare dei test case specifici per ognuno di essi. Questi test case vengono, poi, associati ad un master script che è in grado di invocarli individualmente. Rispetto al modello precedente, la soluzione modulare è sicuramente più scalabile e semplice da mantenere: ogni volta che vengono apportati cambiamenti all'applicazione è possibile identificare i moduli coinvolti e procedere all'aggiornamento dei test case corrispondenti, senza necessità di sconvolgere l'intero framework. Per contro, però, sono necessarie buone competenze di programmazione e il numero di moduli da aggiornare ad ogni modifica dell'applicazione può essere elevato.

3.2.3 Library architecture testing framework

Questo framework rappresenta un'estensione del modello modulare: invece di considerare l'applicazione come un insieme di moduli e produrre test case sulla base di essi si costruisce una libreria di funzioni comuni che possono essere richiamate all'interno degli script di test. Si tratta, quindi, di una soluzione decisamente più scalabile e semplice da mantenere rispetto alle precedenti, grazie alla spinta verso il riutilizzo di funzioni, ma anche più complessa da implementare.

3.2.4 Data-driven testing framework

Questa tipologia di framework è contraddistinta dalla presenza di file contenenti i dati da utilizzare durante l'esecuzione degli script di test. Questi file possono prendere la forma di database, fogli di calcolo, file di testo o file CSV. Queste architetture sono ampiamente utilizzate nel caso di applicazioni che necessitano di essere validate in presenza di grandi quantità di dati di test. Viene particolarmente evidenziata la separazione tra il codice degli script di test e i dati da esso utilizzati (memorizzati separatamente e prelevati durante l'esecuzione, consentendo così di validare milioni di combinazioni iniziali a partire da un singolo script. Affinché questa soluzione sia efficace, però, è necessario investire molte risorse nel mantenimento dei dati di test.

3.2.5 Keyword-driven testing framework

Questi framework, come suggerisce il nome, si basano sulla definizione di un insieme di keyword: delle stringhe che rappresentano delle azioni da eseguire sull'applicazione. Esse vengono memorizzate in apposite tabelle (separate, quindi, dallo script di test), in cui ad ogni keyword vengono associate informazioni come: il nome dell'azione da eseguire, una sua descrizione, l'elemento interessato dall'azione

e il tipo di dato eventualmente richiesto. Tali keyword rappresentano anche l'unità fondamentale di ogni test script, che è ora ridotto ad una sequenza di step. Il framework, quindi, combina test script, tabelle di keyword e file di dati di test in un risultato molto conciso e facilmente comprensibile anche in assenza di particolari competenze tecniche. Questo tipo di architettura presenta elevata scalabilità, le keyword possono essere facilmente riutilizzate per produrre nuovi test case ed è possibile la gestione separata di script, dati e keyword. Tuttavia, si tratta di una soluzione molto complessa che richiede un elevato investimento di risorse iniziale e richiede competenze di programmazione avanzate.

3.2.6 Hybrid testing framework

Un framework ibrido consiste, sostanzialmente, nella combinazione di più architetture con lo scopo di ottenere il meglio da ognuna di esse. Si possono, ad esempio, mettere insieme approcci keyword-driven e data-driven con delle librerie di funzioni comuni o con una metodologia di creazione di test case basata su moduli. In generale, un modello ibrido è ciò che la maggior parte dei framework raggiunge nel corso del tempo.

3.2.7 Test-driven development testing framework

Questo approccio si basa sull'applicazione di principi di Test Driven Development (TDD) ad un test automation framework. I test case vengono, quindi, prodotti prima che l'applicazione venga sviluppata. In questo modo si definisce a priori un chiaro criterio di accettazione del software. L'approccio TDD si è dimostrato di grande beneficio per la qualità del software sviluppato, e usandolo come base per la costruzione di un framework si ottengono risultati come una migliore coverage e una riduzione nel numero di difetti nel codice. Tuttavia, ciò è al netto di un elevato investimento immediato, un rallentamento iniziale dello sviluppo ed un grande sforzo richiesto nel mantenere la test suite.

3.2.8 Behavior-driven development testing framework

L'approccio Behavior-Driven Development è considerato un'estensione del TDD in quanto l'attenzione non è posta semplicemente sui test (quindi sulle necessità dell'utente finale), bensì sulla relazione tra gli sviluppatori e i responsabili del business. Ciò che guida lo sviluppo sono un insieme di scenari che descrivono il comportamento atteso dell'applicazione: essi sono usati come requisiti, test e criterio di accettazione finale, e test automation può essere impiegata per automatizzare questi scenari. Per semplificare la collaborazione tra le personalità tecniche e quelle

di business, vengono utilizzati strumenti che consentono di produrre test scritti in linguaggio naturale.

3.2.9 Creazione di un test automation framework

Quando si sviluppa un test automation framework, le qualità fondamentali da considerare sono scalabilità e manutenibilità: si vuole ottenere un elevato grado di ri-utilizzo di funzioni comuni, una rapida esecuzione dei test e livello minimo di sforzo necessario ad aggiornarli e mantenerli. Inoltre, è importante che il framework sia facilmente espandibile: se la prima risposta alla sua introduzione è positiva, è molto probabile che gli venga richiesto di integrare nuove funzionalità o di dotarsi degli strumenti necessari ad interagire con nuove applicazioni e tecnologie. Di seguito sono presentati i passaggi fondamentali nella creazione di un test automation framework[3]:

1. **Definire i requisiti**: stabilire quali piattaforme/browser/sistemi operativi devono essere supportati, quali sono le fondamentali esigenze di testing e quindi quale parte del processo di test ha la necessità di essere automatizzato. Tutte queste considerazioni vengono raccolte in un documento, il cosiddetto "test automation plan". Esso contiene anche una classificazione dei requisiti in diversi livelli di priorità a seconda delle necessità di business.

2. **Selezionare i tool**: decidere gli strumenti da utilizzare sulla base delle esigenze di test definite in precedenza.

3. **Sviluppare il framework**: lo sviluppo prevede una fase iniziale di analisi dell'architettura dell'applicazione da testare e definizione della struttura del TAF; una fase di sviluppo vero e proprio in cui vengono definiti i dati di test, si assemblano le librerie di funzioni, si producono test script per gli scenari a priorità maggiore e si analizzano sia i risultati prodotti dai test case sia la reportistica generata dal framework (per ottenere un primo feedback sull'efficacia della test automation); una fase conclusiva in cui si definiscono le attività del framework su base continua.

3.2.10 Metriche di test automation

Per poter valutare l'efficacia di una soluzione di test automation e determinare dove si possono apportare miglioramenti, è fondamentale definire delle apposite metriche. Queste possono essere suddivise in tre categorie fondamentali[3]:

1. **Test automation**: l'insieme di metriche relative all'intero processo di testing. Alcuni esempi sono sforzo di creazione, esecuzione e analisi dei risultati dei

test, percentuale di test superati/falliti, percentuale di test automatizzati, tempo di esecuzione dei test, costo dell'automazione

2. **Coverage**: metriche inerenti a quale porzione dell'applicazione è sottoposta a test. Questo concetto fondamentale può essere misurato in termini di requisiti, unit test, regression test, path

3. **Difetti**: essi possono essere misurati ad esempio in termini di densità (quali sono le sezioni più problematiche dell'applicazione), gravità, priorità, causa scatenante, numero di difetti individuati, numero di difetti corretti.

La compilazione di queste metriche può essere affidata a software appositi, che automaticamente acquisiscono le informazioni necessarie e presentano i risultati in forma di report riassuntivi o dashboard.

3.3 Strumenti di test automation

I tool utilizzati per test automation sono scelti in base alla compatibilità con l'applicazione da testare, alla loro flessibilità, al loro costo (sia iniziale sia continuo nel tempo) e al supporto che ricevono. Essi possono essere classificati in diverse categorie, come ad esempio[3]:

1. Strumenti di web testing

2. Strumenti di mobile testing

3. Strumenti di performance testing

4. Strumenti di security testing

Di seguito, sono presentate alcune delle tecnologie e strumenti maggiormente utilizzati nell'ambito della test automation.

3.3.1 Android ADB

Android Debug Bridge è uno strumento basato su linea di comando che consente di comunicare con un dispositivo Android. Di per sé, non si tratta di un tool di test automation ma è ampiamente utilizzata come base per strumenti di automazione di sistemi Android. Digitando il comando di shell adb, si ha accesso ad una shell Unix che permette la comunicazione con il dispositivo Android secondo una struttura client-server. Tale struttura si basa su tre componenti fondamentali[4]:

1. Client: avviato tramite il comando adb sulla macchina di sviluppo, è l'entità che invia comandi al dispositivo Android

2. Daemon: noto come adbd, è un processo in background del dispositivo Android che si occupa di eseguire i comandi adb ad esso inviati

3. Server: entità intermedia tra client e daemon, rappresentata da un processo in background sulla macchina di sviluppo

Nel caso in cui il dispositivo Android sia collegato alla macchina di sviluppo tramite USB, per poter utilizzare ADB è necessario abilitare il debugging USB nelle impostazioni di Android. Tale opzione si trova nel sottomenu Opzioni Sviluppatore. All'avvio del client adb, questo verifica l'esistenza di server adb in esecuzione. In caso negativo, il processo corrispondente al server viene avviato. Esso, quindi, si mette in ascolto sulla porta TCP 5037 per comandi adb client. Dopodichè, il server si connette con tutti i dispositivi Android raggiungibili: viene eseguita una scansione di varie porte (ad esempio, nel caso di emulatori, la scansione è eseguita sulle porte dispari nell'intervallo 5555-5585) e, laddove venga individuato un daemon in esecuzione, viene stabilita una connessione e il dispositivo corrispondente può essere controllato con comandi adb.

- **adb devices**: restituisce una lista dei dispositivi correntemente connessi al server adb, mostrando per ogni dispositivo il suo numero seriale (una stringa univoca generata da adb sulla base della porta a cui il dispositivo è connesso), lo stato (offline nel caso il dispositivo non risponda, no device nel caso non ci siano dispositivi collegati, device nel caso il dispositivo sia connesso al server adb) e, se abilitata con l'opzione -l, una descrizione del dispositivo connesso (utile nel caso di molteplici dispositivi connessi contemporaneamente). Un utilizzo pratico di questo comando è identificare il numero seriale di ciascun dispositivo connesso allo scopo di inviare comandi verso uno specifico sistema Android nel caso in cui vari dispositivi siano connessi contemporaneamente. Ciò viene fatto aggiungendo al comando adb l'opzione -s seguita dal numero seriale ottenuto tramite adb devices.

- **adb install**: installa una APK su un emulatore o un dispositivo connesso. Una variante di questo comando è install-multiple, che installa molteplici APK in una volta. Nel caso venga utilizzato Android Studio, questo comando risulta superfluo poiché l'IDE è in grado di installare app direttamente su emulatori o dispositivi fisici.

- **adb pull**: copia un generico file o cartella dal dispositivo Android alla macchina di sviluppo, dati il suo path di origine e quello di destinazione

- **adb push**:trasferisce una copia di un generico file o cartella della macchina di sviluppo al dispositivo Android, dati il suo path di origine e quello di destinazione.

- **adb kill-server**: termina il server adb. Per poterlo avviare di nuovo, è sufficiente inviare un qualsiasi comando adb.

ADB shell

Questo comandp fondamentale di Android ADB avvia una shell sul dispositivo per interagire con esso. Può anche essere usato per inviare singoli comandi con la sintassi adb shell shell_command. All'interno della adb shell è possibile inviare comandi direttamente all'activity manager (am) per eseguire azioni come istanziare Activity, creare Intent, terminare processi o manipolare la dimensione e la densità di pixel del display. Oltre all'activity manager, adb shell può inviare comandi al package manager (pm) per installare/disinstallare applicazioni e gestire permessi, o al device policy manager (dpm) per controllare la gestione del dispositivo. Altre funzioni eseguibili tramite adb shell sono la cattura di screenshot e la registrazione dello schermo del dispositivo.

3.3.2 Selenium

Selenium è il più diffuso tool di test automation per pagine web. Il suo utilizzo si basa su istanziare un oggetto chiamato WebDriver con cui è possibile inviare comandi al browser e interagire con gli elementi del DOM. Il WebDriver è una API che definisce un'interfaccia indipendente dal linguaggio di programmazione che consente di controllare il comportamento del browser. Per il funzionamento di Selenium sono anche essenziali le implementazioni di WebDriver contenute all'interno di ciascun browser, definite semplicemente come driver. Tali driver hanno il compito principale di gestire la comunicazione tra il browser e Selenium, e normalmente è previsto che la loro implementazione sia affidata allo sviluppatore del browser. Tuttavia, Selenium fornisce dei driver propri per ovviare ai casi in cui ciò non avvenga. Per poter utilizzare Selenium, è necessario installare le relative librerie per il linguaggio di programmazione desiderato, il browser con cui si intende effettuare i test e il corrispondente driver.

Inizializzazione

Prima di eseguire qualsiasi comando è necessario istanziare il driver. Esso è l'oggetto fondamentale necessario ad invocare tutti gli altri comandi. Per fare ciò, è sufficiente invocare il costruttore della classe corrispondente al WebDriver che si intende utilizzare. Per esempio, nel caso il linguaggio di programmazione utilizzato sia Java e si intende usare Chrome come browser, il driver viene istanziato con:

```
WebDriver driver = new ChromeDriver();
```

Una volta creato l'oggetto corrispondente al driver, è possibile navigare verso qualsiasi url utilizzando il metodo "get":

```
driver.get("https://www.polito.it");
```

Raggiunta la pagina web desiderata, è possibile ottenere informazioni dal browser o interagire con gli elementi visibili. Le informazioni ottenibili dal browser includono il titolo della pagina visualizzata (getTitle()), l'url corrente (getUrl()), le dimensioni della finestra (getWidth(), getHeight()) ed eventuali cookie (getCookies()). Selenium consente svariati tipi di interazione con gli elementi a schermo. Tuttavia, per inviare un comando ad un elemento del DOM occorre innanzitutto identificarlo. A tale scopo Selenium offre i metodi findElement e findElements. Tali funzioni richiedono come parametro un selettore, rappresentato da un oggetto appartenente ad una delle sotto-classi di By: una classe astratta le cui implementazioni riflettono svariate possibili strategie di selezione.

Wait timeout

Ogni volta che viene ricercato un elemento con una qualsiasi strategia di locazione, un timer viene utilizzato per stabilire il tempo speso da WebDriver in attesa che l'elemento richiesto venga trovato. Tale timer prende il nome di implicit wait timeout, con un valore di default pari a 0. Oltre a questo esistono altri due timeout[5]: script timeout che specifica la durata massima di esecuzione di script (valore di default pari a 30 secondi) e Page load timeout che limita il tempo impiegato per navigare verso una determinata pagina (valore di default pari a 300 secondi). Questi timer possono essere impostati tramite una richiesta HTTP POST in cui includere un oggetto JSON (timeout configuration) con proprietà script, pageLoad, e implicit e il corrispondente valore di timeout in millisecondi. In pratica, ciò viene eseguito tramite il metodo setTimeouts di driver.manage() specificando il timer che si desidera impostare e il valore corrispondente (insieme alla sua unità di tempo)

```
driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
```

E' comune pratica impostare il timeout di implicit wait (solitamente subito dopo l'inizializzazione del WebDriver) ad un valore sufficientemente alto da accomodare per variazioni dei tempi di caricamento della pagina e durata di eventuali animazioni che accompagnano la comparsa degli elementi di interesse. Infatti, un valore di timeout troppo basso può portare alla generazione di errori di comandi findElement posti subito dopo un comando get o una qualsiasi azione che modifica la pagina corrente. Un'altra azione comune è impostare il timeout a valori particolarmente alti per attendere elementi la cui comparse segue un'azione che può richiedere molto

tempo per essere completata (ad esempio la risposta ad una query da parte di un server)

Selettori

Un selettore è una stringa che può essere utilizzata per identificare elementi di una pagina web in base ai loro attributi o alla loro posizione nella gerarchia del DOM, in modo analogo a come le espressioni regolari possono venire utilizzate per identificare stringhe di caratteri. All'interno di ogni browser è possibile eseguire delle query, fornendo la stringa di un selettore per ottenere in risposta l'insieme degli elementi a cui essa corrisponde. Sono possibili diverse strategie di selezione, ognuna con i propri vantaggi e svantaggi:

- **id**: individuare un elemento a partire dal suo id è la strategia più semplice e più efficiente in termini di prestazioni, in quanto tale attributo è univoco all'interno della pagina web ed è generalmente poco soggetto a variazioni. Tuttavia, sono frequenti i casi in cui l'elemento desiderato non disponga di questo attributo.

- **cssSelector**: individuazione di un elemento attraverso sue caratteristiche come tag, classe, attributi e posizione nel DOM. E' possibile ricercare tag con una specifica classe (mediante una sintassi del tipo tag.classe) o elementi con un valore specifico di un dato attributo (usando una notazione del tipo [attributo=valore]). Selettori di questo tipo sono quelli generalmente raccomandati nel caso l'id non sia disponibile.

- **xpath**: individuare un elemento analizzando il percorso attraverso la gerarchia del DOM che dalla radice conduce ad esso e i valori dei suoi attributi. Questo metodo considera gli elementi di una pagina come file e cartelle di un file system e costruisce selettori nella forma di path. Tali percorsi si presentano come una sequenza di tag separati da "/" e possono essere originati dalla radice o da qualsiasi elemento intermedio (ponendo "//" davanti al tag di partenza). Inoltre, è possibile porre condizioni più complesse sui valori degli attributi, come ad esempio il fatto che il valore dell'attributo di interesse contenga una determinata sottostringa (tale condizione è espressa nella forma "[contains(@attributo,valore)]"). Selettori di questo tipo hanno una maggiore capacità di identificare elementi scarsamente distinguibili ma sono i peggiori in termini di prestazioni e in termini di robustezza del selettore: è sufficiente che un qualsiasi punto intermedio del path venga alterato per rendere il selettore invalido. Infatti, per ridurre al minimo questo rischio è fondamentale che selettori basati su xpath siano di lunghezza minima.

Dopo aver trovato gli elementi di interesse con i metodi Find, è possibile accedere alle loro proprietà ed effettuare asserzioni sul loro valore (ad esempio, verificare che il contenuto di una stringa di testo corrisponda al valore atteso) o interagire direttamente con essi. Per ottenere il testo contenuto in un elemento è possibile utilizzare il metodo getText

element.getText();

che restituisce il testo dell'elemento specificato così come appare nella finestra del browser (o un messaggio appropriato in caso di errore). Questa funzione è anche sfruttata per implementare due ulteriori strategie di selezione elementi specifiche per i link: Link text e Partial link text.

driver.findElement(By.linkText(text))

driver.findElement(By.partialLinkText(text))

I passaggi fondamentali eseguiti da WebDriver sono: l'invocazione di querySelectorAll() passando "a" come parametro; su ciascun elemento trovato, l'estrazione del testo visibile nella finestra (getText()) e l'eliminazione degli spazi presenti all'inizio e alla fine; il confronto della stringa privata di spazi con quella usata come selettore, restituendo gli elementi che hanno una corrispondenza esatta. La selezione per Partial Link text è eseguita in modo analogo, con la differenza che è sufficiente che la stringa di selezione sia contenuta nel testo di ciascun link esaminato.

L'interazione più comune è il click, che può essere facilmente eseguito con il metodo click di WebElement

element.click();

Questo metodo scrolla fino a che l'elemento non sia visibile (nel caso non lo fosse in precedenza) e verifica che elemento sia cliccabile prima di eseguire un click al centro dello spazio da esso occupato, restituendo un messaggio di errore nel caso ciò non sia possibile. Un'altra interazione molto comune è scrivere all'interno di elementi che accettano del testo. Tale azione viene eseguita dal metodo sendKeys

element.sendKeys(string);

Tale funzione scrolla l'elemento fino a renderlo visibile, verifica che sia possibile interagire con esso con azioni da tastiera ed invia la stringa di testo passata come parametro (in caso ciò non sia possibile viene restituito un messaggio di errore). Oltre a normali stringhe di testo, è anche possibile inviare caratteri speciali

attraverso le proprietà statiche della classe Keys (ad esempio Keys.BACK per inviare un backspace e Keys.ENTER per digitare invio).

3.3.3 Appium

Appium è un progetto open-source rivolto a semplificare l'automazione della UI di applicazioni mobile per qualsiasi piattaforma. Appium sfrutta le specifiche del WebDriver di Selenium per esporre una API standardizzata in grado di fornire capacità di automazione specifiche per ogni piattaforma pur rimanendo accessibile da qualsiasi linguaggio di programmazione. Inoltre, Appium eredità la capacità di WebDriver di estendere le proprie funzionalità per fornire supporto nei casi in cui l'interazione utente sia differente tra web e mobile. Siccome le UI di software differenti presentano molte caratteristiche in comune, è possibile utilizzare le specifiche del WebDriver per ricavare primitive (come trovare elementi di UI, visualizzare schermate o scrivere del testo) che possono essere mappate, seppur con qualche differenza, su ogni tipo di piattaforma. Ciò fa risaltare due potenziali problematiche[6]:

1. Una specifica di WebDriver non può essere supportata in quanto non applicabile nel caso di applicazioni mobile (la gestione dei cookie, ad esempio, perde ogni significato al di fuori del contesto del web browser)

2. Una particolare operazione, necessaria nel contesto mobile, non trova corrispondenza con nessuno dei comandi forniti dalla API di WebDriver. In questi casi, Appium si occupa di supportare tale operazione definendo estensioni in linea con lo standard di WebDriver

Il mapping tra specifiche WebDriver e automazione mobile su diverse piattaforme è reso possibile da un modulo software detto Appium driver. Fondamentalmente, un Appium driver è una classe Node.js che estende BaseDriver. Quest'ultima classe racchiude al suo interno l'intero protocollo WebDriver. Un Appium driver, per poter esprimere le proprie funzionalità, deve quindi implementare un override dei metodi Node.js di BaseDriver (che implementano le specifiche originali di WebDriver). Ad esempio, se si vuole implementare un metodo in grado di avviare un'applicazione Android, è sufficiente fare override del metodo che implementa il comando NavigateTo di WebDriver (setUrl):

```
async setUrl(url) {
// do whatever we want here
}
```

Fornendo un deeplink come valore del parametro url è possibile utilizzarlo nel corpo della funzione per avviare un'applicazione. WebDriver, attraverso BaseDriver, fornisce il protocollo di base ma è necessario che le azioni da automatizzare vengano eseguite sulla piattaforma di riferimento (quando si invia un comando per eseguire un tap su un elemento di UI ci si attende lo stesso comportamento del caso il cui il tap venga fisicamente eseguito da un utente). A tale scopo, ogni driver Appium si appoggia alle tecnologie fornite dalla piattaforma di riferimento, ognuna dotata di una propria API. Ad esempio, il driver iOS sfrutta la tecnologia XCUITest sviluppata da Apple mentre il driver Android si interfaccia sia con la tecnologia UiAutomator2 sviluppata da Google sia alle funzionalità rese disponibili da ADB. L'interfaccia con XCUITest presenta un'architettura piuttosto complessa: il framework di Apple richiede che le sue funzioni vengano invocate da codice scritto nei linguaggi Objective-C o Swift. Inoltre, per poter eseguire il codice di XCUITest su un dispositivo è necessario che esso sia in una modalità attivabile unicamente da Xcode o i relativi tool da linea di comando. E', quindi, impossibile implementare una funzione Node.js in grado di eseguire chiamate alla API di XCUITest direttamente. Per ovviare a questo problema, il driver per l'ambiente iOS è stato suddiviso in due moduli: una parte scritta in linguaggio Node.js che è incorporata in Appium e si occupa di gestire i comandi WebDriver, e una parte scritta in Objective-C che viene eseguita sul dispositivo iOS ed esegue direttamente le chiamate alle funzioni di XCUITest. Il modo in cui la comunicazione tra questi due moduli viene gestita è lo stesso protocollo WebDriver. Infatti, la parte scritta in Objective-C è, a sua volta, un'implementazione di WebDriver chiamata WebDriverAgent. Questa separazione permette, in determinate situazioni, al test driver di agire come proxy. Ad esempio, considerando il caso del comando Click Element del driver XCUITest, in alternativa ad implementare il comando il driver può semplicemente inoltrare la richiesta al WebDriverAgent. Quest'ultimo si occuperà, poi, di restituire il risultato al client in modo completamente trasparente al driver. WebDriverAgent è in grado di avviare e terminare applicazioni, eseguire gesture come tap o scroll e verificare la presenza di elementi di UI a schermo. Per utilizzare WebDriverAgent si può ricorrere a qualsiasi bundle UITest, come Xcode e xcodebuild. Per poter avviare uno UITest è anche necessaria un'applicazione diversa da quella che si intende testare in cui il codice delle funzioni di XCUITest viene eseguito. Ad esempio, nel caso in cui si utilizzi Xcode o xcodebuild, tale applicazione viene automaticamente installata sul dispositivo (WebDriverAgentRunner).

XCUITest

XCUITest è un framework di automazione introdotto da Apple a partire dalla versione di iOS 9.3, ed è divenuto l'unico framework supportato da dispositivi Apple a partire da IOS 10. Come menzionato in precedenza, Appium utilizza

WebDriverAgent per connettersi al dispositivo. Tuttavia, nel caso in cui la versione di iOS sia precedente alla 9.3, Appium si appoggia alla libreria UIAutomation. Essa si basa un modulo chiamato bootstrap.js, che viene eseguito dal dispositivo ed è in grado di ricevere i comandi inviati dal client Appium.

UIAutomator2

UIAutomator2 è un framework di automazione test per dispositivi Android, compatibile con applicazioni native, ibride o web based. Esso si basa su due componenti fondamentali: lo UiAutomator2 Driver e lo UIAutomator2 Server. Il primo fa parte dei moduli di Appium ed è utilizzato per inviare comandi al dispositivo; il secondo viene installato sul telefono quando il client Appium avvia una nuova sessione e si occupa di ricevere ed eseguire i comandi inviati dal driver. Per la maggior parte dei comandi, infatti, il driver agisce da proxy e si rivolge al server o ad appium-adb ed altri strumenti della piattaforma Android. Per utilizzare UiAutomator2, oltre ai requisiti base di Appium, è necessario ad esempio[7]:

- Installare Android SDK Platform Tools
- Impostare le variabili d'ambiente ANDROID_HOME e ANDROID_SDK_ROOT
- Installare Java JDK ed impostare la variabile d'ambiente JAVA_HOME
- Abilitare il debug usb sul dispositivo Android o, nel caso si intenda usare un emulatore, installare l'immagine del dispositivo di interesse

Appium Inspector

Appium Inspector è un software che sfrutta un server Appium disponibile per ispezionare le GUI di applicazioni mobile. Sostanzialmente, si tratta di un client Appium con un'interfaccia grafica che semplifica l'utilizzo delle funzionalità di Appium. Pertanto, per poterlo utilizzare, sono necessari un server Appium raggiungibile da Inspector (localhost o un server remoto) e tutti i driver Appium che possono essere necessari. Con Inspector, è possibile specificare le caratteristiche del server Appium a cui connettersi (indirizzo, porta e path) e del dispositivo da ispezionare (le cosiddette desired capabilities).

Desired capabilities

Le desired capabilities sono rappresentate in forma di un oggetto JSON e indicano le caratteristiche del dispositivo e/o applicazione a cui il server Appium deve connettersi. Tali proprietà possono essere generiche o specifiche per una data piattaforma. Alcune delle proprietà generiche sono:

- **platformName**: nome della piattaforma del dispositivo (es. “Android”)
- **platformVersion**: versione del sistema operativo installato sul dispositivo
- **deviceName**: nome del dispositivo
- **udid**: UDID del dispositivo
- **automationName**: nome del driver Appium utilizzato (es. “UiAutomator2” per Android)

Se si considera ad esempio l’ambiente di iOS, si possono definire delle proprietà esclusive come[8]:

- **xcodeOrgId**: Team ID, una stringa univoca di 10 caratteri generata da Apple che identifica lo sviluppatore
- **xcodeSigningId**: Tipo di certificato associato alla Signing Identity, solitamente si utilizza “iPhone Developer”
- **bundleId**: Identificatore del pacchetto che contiene l’applicazione

Mentre in ambiente Android si trovano proprietà come[7]:

- **appPackage**: Identificatore del package dell’applicazione da avviare
- **appActivity**: Identificatore della main activity dell’applicazione
- **appWaitPackage**: Identificatore del primo package utilizzato dall’applicazione
- **appWaitActivity**: Identificatore della prima activity avviata dall’applicazione

Una volta avviata la sessione vengono mostrate un’interfaccia che rappresenta il dispositivo connesso ed il contenuto visibile a schermo su di esso, e la gerarchia dei componenti UI della schermata. Da qui è possibile toccare un qualsiasi punto della GUI dell’applicazione mobile per interagire con essa o evidenziare uno specifico elemento ed analizzare le sue caratteristiche. Al variare dello stato dell’applicazione, la vista mostrata da Appium Inspector non viene aggiornata automaticamente: questa operazione va eseguita manualmente con il pulsante (nome). Un’altra funzionalità fondamentale è quella di cercare componenti della UI usando appositi selettori, specificando il tipo di selettore e la stringa di ricerca. Sono disponibili sia selettori generici (come id e xpath) sia selettori specifici per una determinata piattaforma (come XCUIPredicate nell’ambiente iOS). Inspector, inseriti gli appositi parametri di ricerca, restituisce una lista contenente gli elementi che corrispondono al selettore inserito nell’ordine in cui compaiono nella gerarchia

della UI correntemente visibile. L'utilizzo principale di Appium Inspector, come suggerisce il nome, è ispezionare l'applicazione sotto test: a partire dal flusso di test che si intende automatizzare, analizzare le proprietà degli elementi con cui si vuole interagire in modo da, all'interno del test case, individuarli con selettori robusti ed efficienti ed eseguire le azioni desiderate.

3.3.4 Cucumber

Cucumber è uno strumento di Behavior Driven Development utilizzato per rendere le fasi di esecuzione dei test più facilmente comprensibili a persone che non possiedono competenze di test automation. Esso si basa sulla definizione di cosiddetti feature file (estensione .feature) in cui possono essere contenuti uno o più scenari. Ogni scenario è costituito da un nome e da una sequenza di step: singoli passaggi del test in esecuzione scritti secondo la sintassi Gherkin. Ogni step è rappresentato da una frase scritta in linguaggio naturale che riassume l'azione eseguita da una funzione ad essa associata. A ciascuno step Cucumber richiede l'associazione di un prefisso: una stringa appartenente ad un insieme predefinito ("Given","Then","And" sono alcuni esempi). Tali prefissi non hanno alcuna utilità funzionale: il loro scopo è rendere la sequenza degli step più leggibile. Esistono, tuttavia, alcune convenzioni come, ad esempio, utilizzare il prefisso "Given" per il primo step di ciascuno scenario. La funzione associata allo step realizza nel codice il comportamento descritto descritto a parole nel feature file. La sua definizione è composta da una funzione scritta nel linguaggio di programmazione di interesse e da un'annotazione di Cucumber. Tale annotazione è costituita dal simbolo "@" seguito dal prefisso dello step e dalla stringa utilizzata nel feature file. La funzione può anche dipendere da dei parametri, i quali possono essere forniti dallo step associato alla funzione stessa. Si tratta, in quel caso, di uno step parametrizzato. In fase di definizione dello step, delle sequenze come "tipo" indicano il tipo di ciascun parametro e la sua posizione all'interno della stringa. In caso più di un parametro sia presente, il loro ordine nella signature della funzione segue quello di apparizione nello step. In fase di scrittura del feature file, il valore di ciascun parametro viene definito per poi essere passato alla funzione sottostante in fase di esecuzione. In generale, Cucumber consente di introdurre con sforzo contenuto un ulteriore livello di astrazione che migliora la comprensibilità e semplifica la gestione di un framework di test.

3.3.5 ECU-TEST

ECU-TEST è un software sviluppato da Tracetronic per l'automazione di functional e system testing, in particolare nel contesto dell'automotive software. Il punto di forza principale di ECU-TEST è la possibilità di integrare molteplici strumenti

di test automation per eseguire test end-to-end che coinvolgono molteplici dispositivi (es. Appium e Selenium) in un'interfaccia che consente di sviluppare script (denominati package) senza richiedere competenze di programmazione: è sufficiente selezionare una serie di task predefiniti (detti job), fornire un valore ai loro parametri e collocarli nell'ordine corretto. Per definire una test suite sono necessari 3 tipi di file:

1. **Test configuration file (.tcf)**: contiene le configurazioni di tutti i tool usati dalla test suite
2. **Test bench configuration file (.tbc)**: configurazione generale di ECU-TEST nel contesto della test suite corrente
3. **Package files (.pkg)**: definiscono gli step necessari per l'esecuzione di un test scenario

Ai fini dei test che coinvolgono mobile app e web, i principali tool da configurare sono:

- Appium
- ADB (per sistemi Android)
- Selenium
- Tracetronic Multimedia

Tracetronic Multimedia è un tool fornito da Tracetronic che consente l'elaborazione di video e immagini ottenuti da una camera o memorizzati in un file. Per poterlo utilizzare è necessario configurare il test bench configuration file (fornendo il nome di una porta, il path ad una cartella in cui memorizzare immagini e la lista prioritaria di lingue da utilizzare per OCR) e il test configuration file (indicando la sorgente di video/immagini, sia essa una camera o un file). Multimedia, siccome ECU-TEST è basato su Python, sfrutta la libreria Python OpenCV per fornire funzionalità di image recognition in grado, ad esempio, di identificare testo contenuto in un'immagine o determinare quanto due immagini differiscono tra loro.

3.3.6 Strumenti basati su AI

Sono già stati identificati numerosi campi di applicazione dell'intelligenza artificiale alla test automation. Essi presentano una possibilità di innovazione non dal punto di vista della robustezza dei test e del loro mantenimento, ma anche da quello della generazione dei test stessi. Una delle funzionalità più diffuse tra quelle offerte dalla AI è il cosiddetto self-healing[9], ovvero l'auto-correzione di

errori di test causati da cambiamenti della UI: considerando il contesto web, nei casi in cui test fallisce perché uno specifico elemento non può più essere selezionato, viene eseguita una analisi delle proprietà di ogni elemento della pagina e ad ognuno di essi viene attribuito un punteggio. Esso rappresenta l'affinità rispetto all'elemento desiderato. L'oggetto con il punteggio più alto viene quindi appuntato come nuovo elemento da selezionare durante il test. Funzionalità di questo tipo si, appoggiano, solitamente, ad un database mantenuto nel tempo di elementi identificati durante l'esecuzione di test. Per ciascuno di essi si mantengono informazioni come il tipo, l'insieme degli attributi e il selettore utilizzato per trovarlo. Un'altra funzionalità potenzialmente rivoluzionaria nella generazione di test è il cosiddetto spidering. Strumenti dotati di questa tecnologia sono in grado di esplorare autonomamente la struttura dell'applicazione da testare, raccogliendo informazioni attraverso screenshot, download di HTML o misurazione di tempi di caricamento. Tutte queste informazioni sono raccolte in un data set e utilizzate per il training di un modello che effettua predizioni sul comportamento atteso dell'applicazione. Quindi, tali scenari di utilizzo vengono eseguiti e, se vengono rilevate delle differenze rispetto al comportamento atteso, queste vengono evidenziate come possibile difetto dell'applicazione[10].

Katalon

Katalon è una delle varie piattaforme software disponibili sul mercato che si avvalgono delle funzionalità offerte dall'intelligenza artificiale per fornire un supporto completo al processo di software quality. I software messi a disposizione a questo scopo sono[11]:

- **Katalon TestOps**: uno strumento che consente di pianificare i test, organizzarli in test suite
- **Katalon Studio**: lo strumento fondamentale di creazione ed esecuzione di test
- **Katalon TestCloud**: un ambiente di esecuzione di test basato su cloud

Katalon Studio, in particolare, consente di creare test per applicazioni web e mobile sia attraverso la funzionalità integrata di record and playback sia attraverso un approccio keyword driven. Quest'ultimo offre, a sua volta, due possibili opzioni[11]:

1. Una cosiddetta "manual view" in cui un'interfaccia grafica consente aggiungere nuovi step, riordinarli e specificarne i parametri
2. Una "script view" in cui il test viene visualizzato come uno script tradizionale

Chapter 4

Test Automation presso Reply Concept

4.1 Reply

Reply è una società di consulenza fondata nel 1996 a Torino da Mario Rizzante, Oscar Pepino. Il suo nome originale, Riplai, era un acronimo per "Rizzante, Ignegnatti Pepino Lavorano Ancora Insieme". Ad oggi Mario Rizzante rimane presidente dell'azienda, mentre Oscar Pepino e Tatiana Rizzante (figlia di Mario) svolgono insieme il ruolo di CEO. A partire dal 2006, Reply ha esteso i suoi orizzonti al di fuori dell'Italia ed il suo fatturato, nell'arco tra 2000 e 2021, è passato da 33,3 milioni di euro 1,48 miliardi. I principali ambiti in cui Reply opera sono la consulenza, i sistemi integrati e i servizi digitali. Inoltre, l'azienda si occupa di ideare e sviluppare modelli di business che sfruttano i vantaggi introdotti da paradigmi tecnologici come Big Data, Internet of Things (IoT), Cloud Computing e Intelligenza Artificiale. Tramite la propria consulenza Reply introduce innovazione nei processi dei suoi clienti, li aiuta a definire piani per progetti a lungo termine e rende concreta la loro visione. Uno degli obiettivi futuri di Reply è quello di diventare un punto di riferimento in ambito IoT per poi passare alla cosiddetta Autonomy of Things (AoT). Con questo si intende applicazioni tecnologiche in grado di rendere i computer autonomi nelle loro interazioni con il mondo fisico, senza indicazioni da parte degli esseri umani. Un altro settore su cui l'azienda ha intenzione di puntare è quello della cybersecurity, in particolare su come questa si traduca in protezione dell'individuo. Infatti, come ha anche affermato il Chief Technology Officier (CTO) di Reply Filippo Rizzante, l'obiettivo principale della cybersecurity deve diventare la protezione della persona anziché dei dati o degli oggetti, poiché i dispositivi che diventano bersaglio attacchi possono essere usati come armi per danneggiare fisicamente degli individui. Ciò che distingue Reply da

altre società di consulenza è la sua struttura basata su un insieme di società più piccole, ognuna specializzata nel proprio settore ma tutte allineate con i valori e gli obiettivi della holding. Infatti, tutte le società tendono ad essere molto dinamiche: quando una di esse raggiunge un numero abbastanza grande di persone si tende a dividerla per settorializzare il più possibile. In questo modo si cerca di avere tante società, ognuna specializzata in un determinato campo. Ognuna di esse può, poi, avere al suo interno delle Business Unit. Queste possono essere viste come "mini società", in quanto ogni business unit è a sua volta specializzata in un determinato settore. I principali ambiti di cui l'azienda si occupa sono Financial Services, Logistics, Retail & Consumer Products, Telco & Media, Automotive & Manufacturing Energy & Utilities e Healthcare.

4.1.1 Reply Concept

Reply Concept è una BU di Reply nata nel 2009, a seguito dell'acquisizione del centro R&D Motorola di Torino. Concept nasce come società specializzata nell'IoT (Internet Of Things), con l'obiettivo di servire tutti i clienti con la necessità di sviluppare o testare dispositivi connessi. Con il passare degli anni, Concept ha sviluppato un "Global Test Automation Center" in grado di testare automaticamente, 24 ore su 24 e 7 giorni su 7, gli applicativi o i dispositivi dei clienti, spaziando dai siti web, a centraline di veicoli, dai modem/router delle aziende di telecomunicazioni alle applicazioni per smartphone. Sia durante la fase di sviluppo sia dopo il lancio commerciale, questa BU permette ai suoi clienti di tenere sotto controllo la qualità di prodotti e servizi in modo trasparente. Concept all'inizio contava intorno ai 180 dipendenti (provenienti dal centro di ricerca Motorola) ma ad oggi può fare affidamento su un numero di risorse umane che supera le 300 persone. Di queste, circa il 35% sono "collaboratori esterni". Questi sono dipendenti assunti da una società esterna e successivamente affittati a Concept. In molti casi questo è fatto per via degli alti standard di selezione del personale interno. Concept si è da poco tempo divisa in due società: Concept Quality, che si occupa principalmente di attività di test e validazione di vari sistemi, e Concept Engineering, che si occupa della progettazione e dello sviluppo di sistemi hardware o software (come possono essere applicativi mobile o desktop, siti web, droni, robot). Concept Quality poi a sua volta è suddivisa in più BU. Ad esempio, quella di automotive, che si occupa di acquisire sempre più esperienza sui temi inerenti a questo settore e di svolgere attività di validazione su sistemi di infotainment, centraline telematiche, applicazioni mobile e siti web (sia per quanto riguarda la parte di front end e sia per quanto riguarda la parte di backend). Vi è anche una BU relativa al mondo Telco, che si occupa di gestire i clienti del mondo delle telecomunicazioni, svolgendo attività di validazione su linee internet, telefoniche e apparati hardware come telefoni e modem/router. All'inizio del 2022, la palazzina appartenuta fino a quel momento

a Stellantis (sede storica Fiat dove erano presenti gli uffici di Sergio Marchionne e John Elkann) viene acquistata da Reply e designata come nuova sede di Reply Concept. Sede. Al suo interno è nata "Area 42", un ambiente visitabile dai clienti che mette in mostra le più importanti innovazioni su cui l'azienda sta lavorando. L'Area 42 è suddivisa in sei zone principali[12]:

1. **Autonomous Warehouse Lab:** attraverso veicoli autonomi, droni e piattaforme di Warehouse management evolute, vengono create delle soluzioni autonome per la gestione di operazioni di magazzino quali inventario, picking e operazioni di ottimizzazione;

2. **Last-mile delivery Lab:** si utilizzano soluzioni innovative nell'ambito della logistica last-mile e del manufacturing per l'integrazione e la gestione di Autonomous Vehicles & Robots (AV&R);

3. **Robotics Lab:** in questo spazio vengono testati robot autonomi per lo svolgimento di attività di ispezione e patrolling. I dati raccolti sono poi elaborati ed analizzati tramite cloud;

4. **Connected Products Lab:** sfruttando metodologie Agile e di Continuous Integration & Delivery, vengono progettati e sviluppati dei prodotti connessi con il supporto di acceleratori IoT innovativi;

5. **Blockchain:** uno spazio dedicato all'applicazione della Vehicle Digital Identity definita da MOBI (Mobility Open Blockchain Initiative) per abilitare nuovi processi nell'ambito della mobilità connessa e delle smart cities;

6. **Metaverso:** sperimentando l'integrazione tra tecnologie di AR/VR con servizi di AI in cloud, vengono sviluppate applicazioni immersive che supportano lo svolgimento di attività produttive o di controllo in contesti industriali e consumer.

4.2 Test Automation Framework

L'approccio di Concept Reply alla test automation ha come fulcro la piattaforma chiamata TAF (Test Automation Framework). Essa viene non solo gestita ma anche sviluppata dalla stessa Concept Reply, dando quindi la possibilità di adattarsi alle esigenze del cliente. Infatti, ad ogni cliente viene spesso associata una versione specializzata del TAF. Il TAF rappresenta uno strumento di gestione dei test completo in cui è possibile pianificare l'esecuzione dei test, monitorare in tempo reale e consultare report e statistiche di esecuzione generate automaticamente. Oltre che al personale interno a Reply, viene solitamente consentito l'accesso al TAF anche ai clienti (seguendo opportune politiche di controllo degli accessi) per

questioni di trasparenza. Di seguito sono descritte le funzionalità delle pagine principali che compongono il TAF.

4.2.1 Dashboard

Questa pagina è dedicata alle statistiche di esecuzione dei test. Le metriche di interesse fondamentale sono: percentuale di test superati, percentuale di test falliti e percentuale di test non completati. Tali metriche vengono qui raggruppate sotto varie dimensioni:

- **Daily:** selezionando una data, si possono consultare statistiche riferite a quel giorno (presentate in forma di grafici e istogrammi);

- **Build:** è possibile selezionare una versione di build (il concetto di build viene discusso dettagliatamente in seguito, per il momento lo si può considerare come affine a "test suite") e ottenere informazioni come numero di test eseguiti, test superati rispetto a quelli che compongono la build, test superati/falliti rispetto a quelli eseguiti. Oltre che per la build selezionate, vengono visualizzate statistiche per ognuna delle build più recenti;

- **Feature area:** i test per una specifica applicazione possono essere classificati in base alla funzionalità testata (definita feature area). E' quindi possibile esaminare la percentuale di test superati/falliti per ciascuna feature area;

- **Platform:** vengono analizzate le percentuali di test superati/falliti dal punto di vista di una specifica piattaforma (Web, Android, iOS,...);

- **Device:** vengono visualizzate statistiche riferite all'esecuzione di test su dispositivi Android o iOS;

- **Calendario:** viene mostrato un calendario contenente il numero di test passati e test falliti per ciascun giorno di uno specifico anno;

- **Device per build:** si visualizzano le statistiche riferite all'esecuzione su dispositivi Android o iOS per ciascuna versione di build;

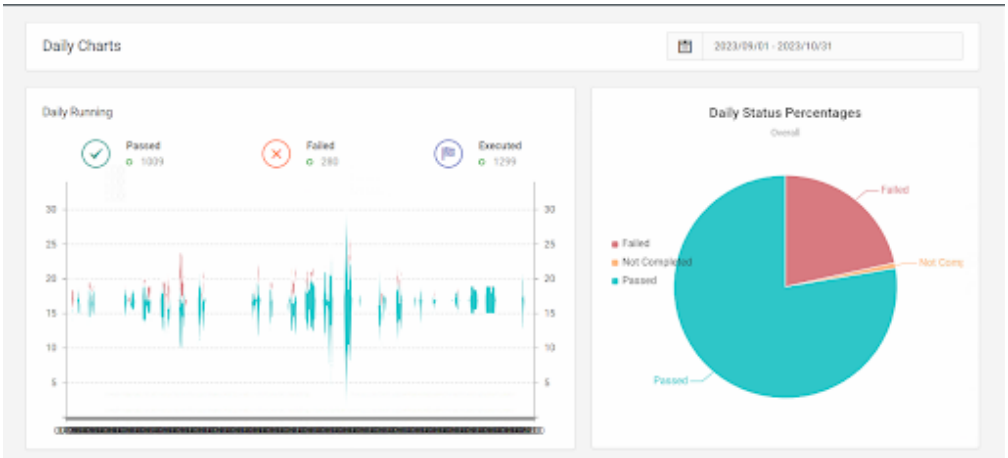

Figure 4.1: Dashboard del TAF Reply

4.2.2 Configuration

In questa pagina si possono configurare le statistiche da mostrare nella dashboard. In particolare si può selezionare:

- quale platform usare per le statistiche riferite ad essa;
- quale insieme di build considerare per statistiche su singola build e complessive;
- quale tipo di device va considerato;
- quali feature area vanno analizzate;
- per il contesto automotive, quale tipo di veicolo bisogna considerare.

Ad ognuna di queste categorie è associata una tabella contenente le possibili opzioni, che sempre da questa pagina possono essere aggiunte, modificate o cancellate.

4.2.3 Test Repository

In questa pagina vengono definite le associazioni tra i test, per come sono definiti all'interno del TAF, e i feature file scritti in linguaggio Gherkin. Ad ogni test sono associate informazioni come:

- **Test Case:** nome del test all'interno del TAF
- **Binaries:** cartella compressa contenente la build di riferimento

- **Command:** parte principale dell'istruzione di esecuzione del test da linea di comando. In particolare, qui viene specificato il file .jar utilizzato
- **Arguments:** argomenti per l'istruzione di esecuzione del test come il driver da utilizzare e, in particolare, il nome del file .feature

Per ciascun test è anche possibile definire degli step (una descrizione passo passo delle azioni compiute dal test, che però non ha alcuna utilità al di fuori del TAF) e visualizzare lo storico delle sue versioni precedenti. All'interno di quest'ultimo è possibile non solo visualizzare data di creazione e autore di ciascuna versione, ma anche riportare il test a una delle sue versioni precedenti. Per semplificare la gestione dei test case, è possibile definire cartelle in cui raggrupparli. Tali cartelle possono, ad esempio, coincidere con le feature area dell'applicazione. La pagina di test repository, oltre a definire un nuovo test case, consente anche di importare o esportare test case in blocco facendo uso di un foglio Excel. All'interno di tale foglio ogni test viene riportato insieme ai parametri descritti in precedenza.

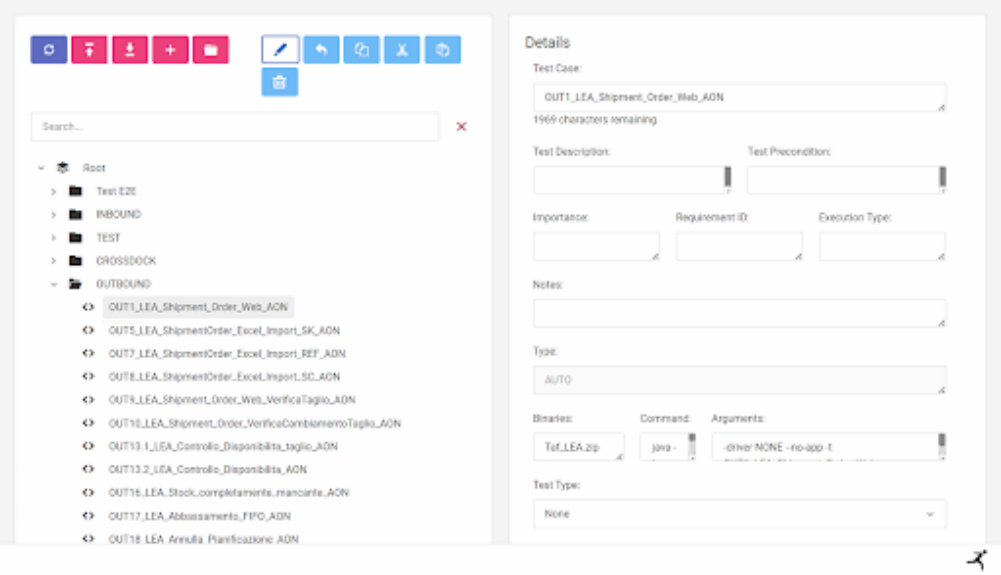

Figure 4.2: Pagina Test Repository del TAF

4.2.4 Execution Progress

Lo scopo di questa pagina è il monitoring in tempo reale di tutti i test in esecuzione. Dopo che viene selezionato un blocco di test da eseguire, è possibile visualizzare quali test sono passati, quali sono falliti, quali sono ancora esecuzione. Inoltre, per ciascun test vengono riportati:

- **Request Time:** timestamp di quando è stata richiesta l'esecuzione del test;

- **Start Time:** timestamp del momento in cui l'esecuzione del test è effettivamente iniziata;
- **Test Name:** nome del test sul TAF;
- **Device:** dispositivo su cui il test viene eseguito
- **Capability:** unione tra Device e sua versione software, usato per tenere traccia dei cambiamenti nei dispositivi di test;
- **Started by:** nome dell'utente che ha avviato il test;
- **Cycle Name:** nome del test cycle (il concetto di test cycle verrà descritto in seguito);
- **Status:** stato di esecuzione del test (superato/fallito/in esecuzione);
- **Progress:** livello di completamento dell'esecuzione del test;
- **Action:** log dell'esecuzione del test da linea di comando. L'output generato da quest'ultima è fondamentale per individuare la causa di eventuale fallimento del test;

4.2.5 Test Schedule

In questa pagina è possibile visualizzare eventuali schedule di esecuzione di test fissate e, se necessario, cancellarle. Per ogni test vengono visualizzate informazioni come:

- **Test Name:** nome del test;
- **Start Time:** timestamp di inizio dell'esecuzione
- **End Time:** timestamp di fine della sessione di test
- **Schedule:** la frequenza con cui il test viene eseguito

4.2.6 Test Execution

Qui vengono definite le schedule di esecuzione di test, dette anche "test cycle". Per ogni test cycle vengono definiti attributi come platform, versione della build, device e feature area. Inoltre, è necessario definire quali test (scelti tra quelli presenti nella pagina di Test Repository) devono essere inclusi nel cycle. Per ogni test aggiunto vengono visualizzate informazioni come stato di esecuzione (passed, failed o no run), l'insieme dei parametri aggiuntivi necessari per l'esecuzione (ad esempio,

il percorso di un file Excel da cui è necessario acquisire dei dati), il timestamp di esecuzione e il nome dell'utente che ha aggiunto il test al cycle. E' possibile selezionare un qualsiasi sottoinsieme di test appartenenti al cycle e programmare l'esecuzione. Per fare ciò necessario selezionare su quale dei dispositivi connessi al TAF eseguire i test e definire una schedule. Essa è caratterizzata da una data di inizio, una data di fine e una frequenza di esecuzione. Quest'ultima può essere definita come intervallo di tempo specificato in minuti o ore oppure come un orario per cui eseguire i test quotidianamente. I test cycle possono essere organizzati in cartelle, e per ogni cartella possono essere visualizzate statistiche sull'esecuzione come numero di test passati/falliti rispetto all'intera cartella e rispetto ad ogni singolo test cycle.

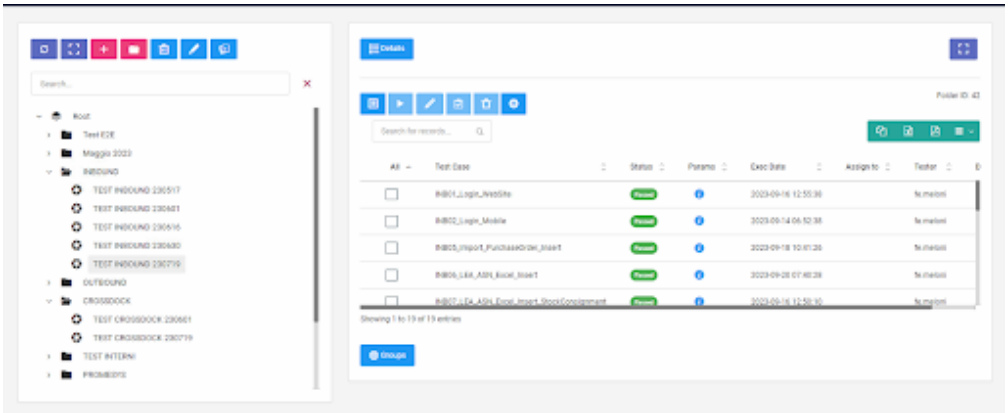

Figure 4.3: Pagina Test Execution del TAF

4.2.7 User Profile

In questa pagina utenti con ruoli amministrativi possono gestire gli account degli altri utenti. Oltre a mostrare la lista con le informazioni personali di tutti gli utenti, viene data la possibilità di crearne di nuovi, modificare gli account esistenti oppure disabilitarli. A seconda dei casi, possono essere assegnati account con permessi amministrativi ad utenti esterni a Reply, permettendo loro creare altri profili utente se lo ritengono necessario.

4.2.8 Roles Management

Questa pagina è dedicata alla gestione dei ruoli assegnabili a ciascun utente del TAF, da quelli amministrativi come ADMIN e SUPERADMIN a quelli operativi come TESTER. Ad ogni ruolo sono assegnati un nome, una breve descrizione, una data di creazione e un insieme di diritti. Tali diritti definiscono le azioni

consentite a ciascun ruolo all'interno del TAF. Una politica adoperata per il TAF di un particolare cliente è la seguente:

- **SUPERADMIN:** accesso e visibilità totale;
- **ADMIN:** controllo totale degli utenti, salvo la possibilità di cancellarli; accesso completo a configuration, test repository, test execution, test planning e dashboard; possibilità di visualizzare ruoli inferiori e assegnare loro privilegi. Non sono consentiti l'accesso ai log (descritti in seguito) e alla pagina di gestione dei gruppi di utenti;
- **TESTER:** accesso in sola lettura alle configurazioni; possibilità di creare ed eseguire nuovi test cycle; accesso alla dashboard e alla test schedule; accesso alla pagina dei ruoli con possibilità di assegnare privilegi;
- **USER:** possibilità di visualizzare e modificare le configurazioni; accesso completo a dashboard, test repository, test execution e test schedule. Non sono consentiti l'accesso a log, gestione degli utenti, gestione dei ruoli e gestione dei gruppi;
- **VIEWER:** accesso in sola lettura alle pagine di dashboard, test repository e test execution.

4.2.9 Groups Management

A seconda del progetto, può essere necessario definire degli insiemi di utenti a cui viene garantito accesso esclusivo ad alcune risorse. Ad esempio, si può desiderare che solo gli utenti interni a Reply abbiano accesso ad un determinato insieme di test. Per fare ciò è possibile definire un gruppo (detto anche "region"), assegnare ad esso dei test (ad esempio, attraverso la pagina di test execution) e rendere tali test visibili ai soli membri del gruppo (con l'eccezione degli utenti SUPERADMIN). La pagina Groups Management consente, agli utenti autorizzati, di visualizzare l'elenco dei gruppi creati. Ogni gruppo è definito da un nome, una descrizione ed una data di creazione. Da questa pagina è anche possibile, per ogni gruppo, aggiungervi utenti, modificarne le relative informazioni ed eliminarlo.

4.2.10 Logs

In questa pagina, solitamente ad uso esclusivo degli utenti con ruolo di SUPERADMIN, è possibile consultare l'elenco delle azioni eseguite sul TAF da ogni utente. Le azioni visualizzate possono essere filtrate selezionando un opportuno intervallo di date ed includono informazioni come il nome dell'utente, il tipo di azione (es. LOGIN) con un messaggio associato (es. "User has logged in") e il timestamp in

cui ogni azione è stata effettuata. Questo elenco può essere, se necessario, esportato in formato Excel, PDF o CSV.

4.2.11 Mail

Questa pagina contiene il riepilogo di tutte le mail inviate automaticamente dal TAF (ad esempio, ogni volta che viene creato un nuovo utente). Per ogni mail vengono visualizzati la data, l'oggetto e il contenuto.

4.3 TAF Build

Precedentemente, nel contesto della pagina di Test Repository, si è introdotto il concetto di "build". Essa consiste in una cartella che contiene tutti i test definiti per una specifica versione del TAF, i file necessari per il loro avvio e l'insieme di report di esecuzione generati. Il processo di sviluppo di test consiste, fondamentalmente, nell'utilizzo di Cucumber per produrre file .feature in linguaggio Gherkin, dove ogni step sono associate funzioni scritte in linguaggio Java. Tutto ciò che concerne l'esecuzione di test, invece, è basato sull'utilizzo di TestNG(un framework di testing su Java basato su JUnit) e Automix(una libreria di test automation che si occupa di gestire i driver, le modifiche fatte alle funzionalità di base di TestNG e le annotazioni di BeforeScenario e AfterScenario non definite da Cucumber. Di seguito sono presentati i file/cartelle più importanti che compongono una build.

4.3.1 features

All'interno di questa cartella sono contenuti tutti i file con estensione .feature che descrivono i test case. Infatti, questi sono scritti in linguaggio Gherkin attraverso l'uso di Cucumber. Ogni feature è caratterizzata da un nome, da uno o più scenari (sequenze di step Cucumber) e, all'occorrenza, da un insieme di annotazioni associate azioni da compiere prima o dopo l'esecuzione del test (introdotte dall'espressione @Before o @After). Ad esempio, vengono spesso inclusi dei cosiddetti "After scenario" con lo scopo riportare l'applicazione allo stato originale al termine dell'esecuzione del test (facendo logout, annullando modifiche fatte ad un profilo utente,...).

4.3.2 src

In questa cartella sono contenuti tutti i file Java. Essi sono suddivisi a loro volta in due sottocartelle:

1. main/resources

2. test

La cartella main/resources contiene, ad esempio, file Excel da cui vengono recuperati dati di test oppure oggetti JSON che definiscono parametri di connessione ad alcune API. La cartella test contiene le definizioni di tutti gli step Cucumber. Tali definizioni prendono la forma di metodi statici all'interno di classi Java che raccolgono tutti gli step di una specifica piattaforma (Android, iOS, Chrome,...). All'interno di queste classi il driver è assegnato ad una proprietà statica. In questo modo, esso diventa accessibile come unica istanza sia ai metodi interni sia ad altre funzioni esterne che potrebbero necessitare del driver. E', infatti, necessario evitare che esso venga ri-istanziato durante l'esecuzione del test. Per associare il metodo Java ad uno step Cucumber viene aggiunta una annotazione, in cui si specifica la stringa in linguaggio Gherkin da utilizzare nei file .feature. Nel caso in cui lo step accetti dei parametri, questi sono indicati dalla presenza di parentesi graffe in cui viene indicato il tipo del parametro. Esso ottiene una corrispondenza nella definizione del metodo Java associato

```
1 @Then("I open the {string} iOS app for the first time")
2     public static void openApp(String appName) {
3         Utils.driverType = DriverTypes.IOS;
4         driver = IOS.openApp(appName);
5         driver.manage().timeouts().implicitlyWait(TIMEOUT_TIME,
    TimeUnit.SECONDS);
6         CommonPage.loadDriver(DriverType.IOS, driver);
7     }
```

La maggior parte di questi metodi, tuttavia, si comporta come un semplice wrapper e si occupa di richiamare la funzione contenente l'effettiva implementazione dello step. Infatti, oltre alla cartella contenente gli step esiste un'altra sottocartella (denominata "functions") in cui sono contenute le classi che interagiscono direttamente con le funzioni del driver. Anche in questo caso, viene definito un file .java per ciascuna delle piattaforme da testare o per applicazioni che necessitano di funzioni specifiche. Quello che si ottiene è una corrispondenza come Android.java e AndroidSteps.java. All'interno della cartella functions è anche presente un file detto Utils.java. La classe Utils contiene, oltre ad alcune funzioni di supporto come metodi per la manipolazione di stringhe, l'implementazione delle funzioni da eseguire prima o dopo l'esecuzione di un test (i cosiddetti BeforeScenario e AfterScenario) e una HashMap (chiamata "variables") utilizzata per memorizzare singoli dati prodotti dagli step. Quest'ultima è fondamentale, ad esempio, se si vogliono passare dei dati dinamici da uno step all'altro. Un altro file molto importante contenuto nella cartella test è TestRunner.java. Questa classe si occupa di passare gli argomenti provenienti da launch.json ad Automix, creare una nuova istanza di TestNG ed avviare il test.

4.3.3 launch.json

Questo file viene utilizzato per l'esecuzione di test all'interno dell'editor di codice (Visual Studio Code). Come suggerisce l'estensione, launch.json contiene un insieme di oggetti JSON che definiscono le configurazioni dell'IDE per l'esecuzione di ciascuna feature attraverso il debugger Java. Ognuna di tali configurazioni specifica:

- **type:** tipo di configurazione(in questo caso "java");
- **name:** nome della configurazione(coincidente con il nome della feature)
- **request:** tipo di richiesta (impostata a "launch" poiché si vuole avviare una nuova istanza
- **mainClass:** nome completo della classe Java (in questo caso, "ta.TestRunner")
- **projectName:** nome del progetto in cui il debugger deve cercare classi (lasciato vuoto)
- **args:** array di argomenti per l'esecuzione della feature
- **env:** array di variabili d'ambiente eventualmente necessarie per l'esecuzione del test

```
{
      "type": "java",
      "name": "2.2_SocialLoginFacebookF2M",
      "request": "launch",
      "mainClass": "ta.TestRunner",
      "projectName": "",
      "args": [
        "-driver",
        "No Driver",
        "-t",
        "2.2_SocialLoginFacebookF2M"
      ],
      "env": {
        "EMAIL": "testReply503@mailinator.com",
        "ArduinoPort": "COM3",
      }
    }
```

4.3.4 config.properties

Questo file viene utilizzato per la definizione di proprietà necessarie all'esecuzione di test per dispositivi mobile (le cosiddette desired capabilities). Tali informazioni vengono recuperate nel momento in cui viene inizializzato un driver di tipo Android o iOS, in modo da potersi connettere al dispositivo.

4.3.5 test-output/reports

Questa cartella è dedicata ai report di esecuzione. Ogni report viene generato automaticamente come cartella compressa. Essa contiene, a sua volta:

- un file JSON che riporta il risultato dell'esecuzione di ciascuno step Cucumber;
- un log di tutto ciò che Automix ha stampato su linea di comando durante il test (come timestamp di esecuzione di ciascuno step o dettagli di eventuali eccezioni);
- una copia del file config.properties utilizzato durante il test
- una cartella di screenshot ottenuti in seguito all'esecuzione di ciascuno step. Questi vengono sempre acquisiti se non viene aggiunta l'annotazione "@Screenshot(disabled = true)";
- un insieme di file JSON che contengono le risposte a richieste http eventualmente inviate dal test;
- una pagina HTML riassuntiva in cui vengono mostrati lo stato di esecuzione di ciascuno step e, in particolare, la causa di fallimento degli step non superati.E' presente, inoltre, una dashboard in cui è possibile osservare il numero di scenari (test) superati contro quello quello di scenari falliti, il totale di step superati contro quello di step falliti e la durata complessiva di esecuzione della feature.

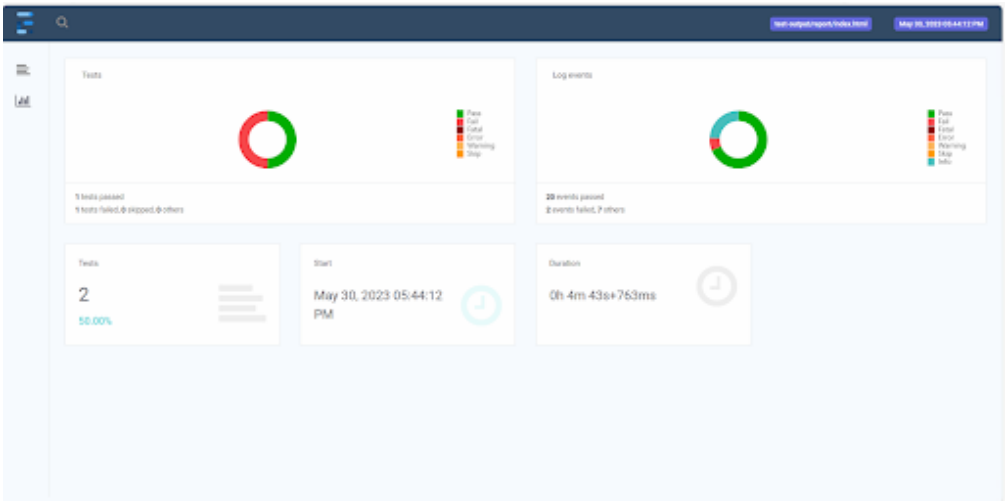

Figure 4.4: Esempio di report

4.4 Esempi di test

In seguito vengono mostrati degli esempi di come l'architettura descritta in precedenza sia in grado di integrare diverse tecnologie e di come i driver di Selenium e Appium possano automatizzare alcuni scenari di utilizzo più complessi. Alcuni dettagli saranno necessariamente omessi poiché i SUT(Software Under Test) da cui gli esempi seguenti vengono tratti sono protetti da NDA.

4.4.1 Integrazione con Arduino

L'applicazione in questione consente gestire le operazioni di ricarica di un'auto elettrica presso una specifica colonnina. Essa può, in ogni momento, trovarsi in uno di 3 stati: scollegato(A), ready(B) e carica(C). Lo stato corrente può essere controllato attraverso una manopola posta sulla colonnina. Per automatizzare una procedura completa di ricarica, viene utilizzato un servomotore connesso ad una scheda Arduino. Inviando i comandi appropriati a quest'ultima, è possibile azionare il servomotore e portare la manopola sulla colonnina nella posizione desiderata. Per fare ciò, vengono definite due classi Java (Arduino.java e ArduinoSteps.java) che definiscono le funzioni necessarie a connettersi/disconnettersi alla scheda e manovrare il servomotore. Per potersi connettere ad Arduino, oltre ad un collegamento USB con il computer su cui viene eseguito il test, è necessario identificare il numero di porta COM della scheda (nei dispositivi Windows, ciò può essere fatto accedendo alla finestra di "Gestione dispositivi"). Esso varia da un dispositivo all'altro, perciò è essenziale verificarlo ogni volta che si tenta di connettersi ad Arduino da un dispositivo diverso. Una volta stabilito il numero di

porta, l'informazione viene resa disponibile al test attraverso le variabili d'ambiente di launch.json. La funzione che si occupa della connessione (startConnection) usa questo dato per avviare la scheda, e attende che l'inizializzazione di Arduino sia completata. Quando la connessione viene stabilita con successo, si ottiene un'istanza della classe IODevice, che funge da interfaccia per inviare comandi ad Arduino.

```
public static void startConnection()
            throws IOException {
        try {
            String myPort = getEnvironmentVariable("ArduinoPort"); //
    modify for your own computer & setup.
            try {
                currentBoard = new FirmataDevice(myPort);
                currentBoard.start(); // start comms with board;
                System.out.println("Board started.");
                ReportFormatter.logInfo("Board started");
                currentBoard.ensureInitializationIsDone();
            } catch (Exception ex) {
                System.out.println("couldn't connect to board.");
                Assert.fail("couldn't connect to board", ex);
            }
        } catch (Exception e) {
            System.out.println("Dind't find ArdinoPort variable on
    the node config");
        }
    }
```

Il servomotore è connesso alla scheda attraverso un pin predefinito. La funzione che si occupa di controllare il movimento ottiene una referenza ad un oggetto di classe Pin, un'interfaccia verso il pin di interesse e la usa per impostarlo in modalità "Servo". Quindi, il valore di output del pin viene impostato in modo da coincidere con l'angolo da far assumere al servomotore. Sono definiti degli step Cucumber per passaggio a ciascuno dei 3 stati (A,B,C), ed ogni step imposta il servomotore ad un angolo predefinito (0° per lo stato A, 30° per lo stato B, 60° per lo stato C).

```
    public static void servoMove(Integer angle, Integer pin) throws
    IllegalArgumentException, IOException {
        Pin myLED = currentBoard.getPin(9);
        // myLED.setValue(0);
        myLED.setServoMode(0, 180);
        myLED.setValue(angle);
        ReportFormatter.logInfo("Angle setted to " + angle);
        try {
            Thread.sleep(1000);
```

```
9           } catch (Exception ex) {
10              System.out.println("sleep error.");
11          }
12      }
```

Al termine del test, viene eseguito un AfterScenario che riporta il servomotore nella posizione A (anche nel caso un'eccezione abbia interrotto l'esecuzione della feature) e termina la connessione con Arduino.

4.4.2 Gesture e image recognition

All'interno della stessa applicazione è possibile ricercare le colonnine di ricarica e visualizzare i risultati su una mappa. Ai fini di test, si imposta la ricerca di una colonnina predefinita. Analizzando i pin corrispondenti ai risultati di ricerca con Appium Inspector emerge che presentano i medesimi attributi, quindi occorre un metodo alternativo per identificare in modo consistente il pin desiderato. La soluzione adottata è un processo di image recognition: viene predisposto un file .jpg contenente l'elemento da trovare; questo viene aperto e il corrispondente flusso di byte viene convertito in una stringa in base 64; questa viene, quindi, passata come parametro ad una particolare variante del metodo find del driver Appium (findElementByImage). Questa funzione si basa su OpenCV, una libreria open-source di computer vision che offre molteplici funzionalità tra cui processing di immagini, analisi di video e identificazione di oggetti, per cercare un elemento identificato da un'immagine all'interno di uno screenshot dello schermo corrente. Infatti, l'utilizzo di findElementByImage richiede che sia la libreria sia il suo plugin per Node.js siano installati sulla macchina di test. Se necessario, è possibile impostare parametri come la tolleranza nel riconoscimento dell'elemento.

```
1       public static void clickOnElementByImageIOS(IOSDriver<
    MobileElement> driver, String imageName) throws IOException {
2           try {
3               File img = new File("src/test/img/" + imageName);
4               String base64EncodedImageFile = Base64.getEncoder().
    encodeToString(Files.readAllBytes(img.toPath()));
5               MobileElement element = driver.findElementByImage(
    base64EncodedImageFile);
6               System.out.println("----------FOUND IMAGE----------");
7               System.out.println(element.getId());
8               System.out.println("-------------------------------");
9               element.click();
10          } catch (Exception e) {
11              e.getMessage();
12              Assert.fail("The image is not present", e);
13          }
```

```
14      }
```

Per poter trovare il pin desiderato, però, è necessario che questo sia sufficientemente visibile a schermo. E' necessario, quindi, navigare all'interno della mappa attraverso l'automazione di gesti come swipe e pinch. In questo caso, è possibile osservare delle differenze tra Android e iOS. L'operazione di swipe è analoga tra le due piattaforme: per scorrere lo schermo è necessario definire una catena di azioni, istanziando un oggetto di classe TouchAction (il cui costruttore riceve l'istanza corrente del driver come parametro). Vengono quindi concatenati i metodi seguenti:

1. **press(startPoint):** si avvia il gesto indicando un punto dello schermo in cui premere. Quest'ultimo è identificato da un oggetto di classe PointOption, che viene costruito a partire dalle coordinate xy del punto (solitamente, viene scelto il centro dello schermo);

2. **waitAction(wait):** si attende per un determinato lasso di tempo prima di procedere. Questo è rappresentato da un'istanza di WaitOptions costruita a partire dal tempo di attesa desiderato;

3. **moveTo(endPoint):** esegue il movimento dal punto di contatto corrente alla destinazione passata come parametro. Questa è definita in modo analogo al punto iniziale;

4. **release():** rimuove il tocco corrente dallo schermo (rappresenta l'azione di sollevare il dito)

5. **perform():** esegue la sequenza di azioni specificata in precedenza;

```
1       new TouchAction(driver)
2                       .press(pointOptionStart)
3                       .waitAction(WaitOptions.waitOptions(Duration.
    ofMillis(PRESS_TIME)))
4                       .moveTo(pointOptionEnd)
5                       .release().perform();
```

Per quanto riguarda l'azione di pinch, questa viene eseguita seguendo un approccio diverso rispetto al precedente: invece di definire una catena di azioni, ci si appoggia a degli script predefiniti. Essi, scritti in linguaggio Javascript, fanno parte dei backend di UiAutomator2 (Android) e WebDriverAgent (iOS). Nel caso di Android, lo script si chiama pinchOpenGesture e si basa sulla definizione di un'area di pinch (una porzione rettangolare dello schermo) in cui eseguire uno zoom. Sono richiesti come parametri:

- **left:** offset dell'estrema sinistra dell'area di pinch rispetto allo schermo;
- **top:** offset dell'estremo superiore dell'area di pinch rispetto allo schermo;
- **width:** larghezza dell'area di pinch;
- **height:** altezza dell'area di pinch;
- **percent:** percentuale dell'area di pinch da coprire durante l'azione. Un valore pari a 100% implica il massimo dello zoom in;

Specificando l'id di un elemento anziché le informazioni sull'area di pinch, è possibile eseguire l'azione direttamente su di esso. Esiste anche una controparte di questo script, pinchCloseGesture, che funziona in modo analogo ma esegue uno zoom out. Nel caso di iOS, lo script prende il nome di "pinch". In questo caso, i parametri fondamentali sono:

- **scale:** scala del pinch, ovvero l'indicatore di quali debbano essere la direzione e l'intensità dello zoom. E' espresso tramite un valore float (se compreso tra 0 e 1 si ottiene uno zoom out; se maggiore di 1 si ha uno zoom in);
- **velocity:** velocità dell'azione di pich, espressa in termini di fattori di scala al secondo;

Specificando come ulteriore parametro l'identificatore di un elemento, è possibile concentrare l'azione di pich su di esso anziché l'intera applicazione. Per eseguire lo script, è necessario abilitare il driver all'esecuzione di Javascript. Per fare ciò si utilizza direttamente quest'ultimo per istanziare un oggetto di classe JavascriptExecutor. A questo punto, è sufficiente invocare il metodo executeScript e passargli il nome dello script da eseguire ed una Map dei parametri ad esso associati.

```
1    JavascriptExecutor js = driver;
2    js.executeScript("mobile: pinchOpenGesture", params);
```

4.4.3 Test di messaggi CAN

Nell'ambito dei servizi connessi su veicolo, si vuole testare il flusso di segnali da rete CAN fino al client di un'applicazione mobile che mostra lo stato del veicolo (passando attraverso un backend che acquisisce e rielabora i dati). Lo scenario di test prevede l'utilizzo di una test bench, che rappresenta una rete CAN priva di centraline, e un sistema di simulazione di messaggi CAN (basato sull'utilizzo di CANAlyzer). La test bench ha il compito di simulare il veicolo: si mette in

ascolto dei segnali che viaggiano su rete CAN e li invia ad un apposito backend. CANAlyzer è un software sviluppato da Vector che viene utilizzato per l'analisi e la simulazione di segnali CAN. L'insieme di segnali da inviare è descritto in file .dbc, in cui ognuno di essi è associato al messaggio a cui appartiene (un messaggio è composto da uno o più segnali) e la stringa di bit corrispondente. Per comunicare con la test bench, è necessario un sistema hardware (associato a CANAlyzer) detto CAN Case. Esso svolge la funzione di interfaccia tra la macchina di test e la rete CAN. Ai fini del test, quindi:

1. a partire da un file Excel che descrive i messaggi associati ad ogni scenario, si identificano i segnali corrispondenti all'interno del file .dbc;
2. i segnali vengono inviati alla rete CAN attraverso CANAlyzer e il CAN Case;
3. Il traffico che viaggia sulla rete viene intercettato ed inviato ad un backend, dove viene processato;
4. Attraverso chiamate a delle apposite API, si verifica che i segnali siano stati correttamente generati e trasmessi;
5. All'interno dell'applicazione, si verifica che lo stato del veicolo visualizzato sia coerente con i segnali CAN;

Per la gestione del file Excel vengono definiti degli appositi step Cucumber (gestiti dalle classi ExcelSteps.java e Excel.java) per leggere e salvare il contenuto di una tabella, filtrare le righe in base al valore assunto da una determinata colonna (fornito da una variabile d'ambiente) e aggiungere nuove righe. Dopo che le informazioni del foglio di calcolo sono state acquisite, si identifica la riga corrispondente al test da effettuare. Da quest'ultima vengono estratti i messaggi CAN corrispondenti, che attraverso CANAlyzer e il CAN Case sono inviati al test bench e, da lì, inviati a backend. Quindi, viene periodicamente inviata una richiesta ad una specifica API del backend fino a quando si ottiene la risposta desiderata o si attende oltre un determinato intervallo di tempo (a significare il fatto che si sia verificato un errore). Se il server fornisce il tempo la risposta attesa, si procede al test dell'applicazione per verificare i messaggi siano elaborati e presentati correttamente.

Chapter 5

Possibili migiloramenti e alternative

Da ciò che è stato presentato in precedenza su come Concept Reply pratichi test automation si possono trarre le seguenti conclusioni:

- La piattaforma TAF presenta una struttura ibrida e, grazie al fatto di essere sviluppata internamente, possiede un'elevata flessibilità. E' in grado non soltanto di adattarsi facilmente ad esigenze di testing differenti ma anche di integrare diverse tecnologie;
- L'approccio BDD facilità la comunicazione verso i responsabili di business e consente un elevato livello di trasparenza verso il cliente;
- Molto tempo viene dedicato alla scrittura di feature e step Cucumber, ed è richiesto uno sforzo continuo per il mantenimento della piattaforma e delle test suite

Si possono, quindi, cercare delle soluzioni che consentano di ridurre lo sforzo richiesto per mantenere il TAF o di velocizzare il processo di creazione di test, siccome l'obiettivo è quello di minimizzare il time-to-market.

5.1 Strumenti di AI (Katalon)

All'interno del TAF non viene fatto uso di strumenti basati sull'intelligenza artificiale. Come mostrato in precedenza, essi possono semplificare di molto le operazioni di produzione di test attraverso generazione automatica e/o auto-riparazione. Una possibile soluzione potrebbe essere, ad esempio, la gestione dei test attraverso la piattaforma di Katalon. Katalon Studio, come descritto in precedenza, offre la

possibilità di registrare le azioni eseguite dall'utente sull'applicazione e riprodurre la sequenza di interazioni in ogni momento.

5.1.1 Configurazione della registrazione

Per registrare su web è sufficiente cliccare l'opzione "Record Web", selezionare l'URL a cui navigare e quale browser utilizzare. Katalon Studio fornisce autonomamente dei WebDriver per Chrome, Microsoft Edge, Modzilla Firefox e Internet Explorer. Questi possono essere anche individualmente aggiornati attraverso la sezione Tools/Update WebDrivers o sostituendoli manualmente nella cartella di Katalon Studio "configuration/resorces/drivers". Per registare su mobile, invece, è necessario installare separatamente Appium e specificare il suo path all'interno delle "Katalon Studio Preferences" (accessibili tramite la sezione "Window"), in particolare nel tab "Katalon/Mobile". Selezionado "Record Mobile" e specificando il tipo di dispositivo (Android o iOS) si apre la finestra "Mobile Recorder" in cui è necessario specificare informazioni come:

- **Device Name:** nome del dispositivo (Katalon fornisce una lista di dispositivi connessi tra cui scegliere;
- **Start with:** qui si dichiara se l'applicazione mobile da lanciare viene identificata attraverso Application File (percorso per il file .apk per Android o .ipa per iOS) o Application ID (nome del package di un'app Android o identificatore del bundle di un'app iOS).

Dopo che la registrazione viene avviata si visualizzano:

- una "Device View" che mostra il contenuto a schermo del dispositivo mobile);
- un elenco di tutti gli elementi che compongono la UI, organizzati secondo la loro gerarchia (All Objects);
- Un insieme di azioni fondamentali che possono essere eseguite sugli elementi della UI (Available Actions) come tap, swipe o scrivere del testo

5.1.2 Registrazione

Vengono registrati ogni azione eseguita sull'applicazione e tutti gli elementi che compaiono sulla UI. Nella tabella "Recorded Actions" ogni interazione con l'utente viene memorizzata in termini di tipo di azione eseguita, elemento con cui l'interazione è avvenuta (ad esempio un bottone) ed eventuali input e output. In ogni momento questa tabella può essere manualmente modificata aggiungendo, rimuovendo, riordinando e alterando le azioni già presenti. Uno degli utilizzi principali di questa

funzionalità è, ad esempio, l'aggiunta di step di verifica di presenza di determinati elementi della UI. In un'altra tabella, detta "Captured Objects", vengono memorizzate informazioni su tutti gli elementi di UI con cui si è interagito durante la registrazione. Ad ognuno di essi vengono associati automaticamente un nome, una strategia di locazione e un corrispondente locatore (in grado di identificare l'elemento in modo univoco). E' anche possibile scegliere una diversa strategia di locazione e generare, quindi, un nuovo locatore. Le strategie supportate includono la maggior parte di quelle supportate anche da Appium, come Accessibility ID, Class name, ID, Xpath e Image. Nel caso di test mobile è possibile, in ogni momento durante la registrazione, selezionare "Capture Object". Questa azione serve ad aggiornare la Device View con la schermata correntemente visibile sul dispositivo e la lista All Objects con la gerarchia di elementi corrispondente.

Spy

Oltre alla funzione di "Record" è possibile ricorrere alla funzione di "Spy". In questo caso, non viene memorizzato uno script, bensì ci si limita all'acquisizione di oggetti. Su pagine web, se si sposta il cursore su di un elemento questo viene evidenziato e, premendo il tasto destro del mouse, viene mostrata l'opzione per salvarlo. In ambito mobile, data una schermata, è possibile selezionare gli elementi di UI desiderati dalla Device View o dalla lista All Objects e salvarli nel repository.

5.1.3 Manual View e Script View

Quando si interrompe la registrazione, viene data la possibilità di modificare un'ultima volta le azioni registrate prima che lo script corrispondente venga salvato. Inoltre, è possibile scegliere quali degli oggetti catturati salvare all'interno dell' "Object repository". Esso consiste in un database in cui vengono memorizzate le informazioni di tutti gli oggetti catturati dai test di una determinata test suite. Il repository può essere organizzato in cartelle, in cui ogni oggetto viene identificato in modo univoco dal proprio nome. Ogni cartella può corrispondere, ad esempio, ad una piattaforma o ad una particolare applicazione. Quando si devono salvare nuovi oggetti è possibile specificare la cartella di destinazione e, in caso di conflitto, scegliere se rimpiazzare gli elementi preesistenti, creare dei duplicati o unire le modifiche agli oggetti già presenti. Una volta che il test case è stato salvato è possibile visualizzarlo in Manual View, in cui è possibile modificare liberamente l'elenco di step presenti, o in Script View, in cui ciascuno step è descritto da una funzione Groovy come:

```
WebUI.click(findTestObject('Object Repository/a_Bestseller'))
```

L'approccio utilizzato nella creazione di script è di tipo keyword-driven: Katalon mette a disposizione una serie di keyword predefinite, raggruppate in package come "WebUI", "Webservice" o "Mobile". Come si può osservare, la ricerca di elementi non utilizza direttamente dei locatori bensì si rifà alle informazioni contenute nell'Object Repository. In questo modo, oggetti e test cases hanno la possibilità di evolvere in modo indipendente.

5.1.4 Playback

I test case salvati possono essere, quindi, eseguiti in qualunque momento. Ad esecuzione terminata è possibile visualizzare il risultato di ciascuno step, l'output prodotto su linea di comando (all'interno della sezione "Console") ed uno storico dei risultati di test run precedenti.

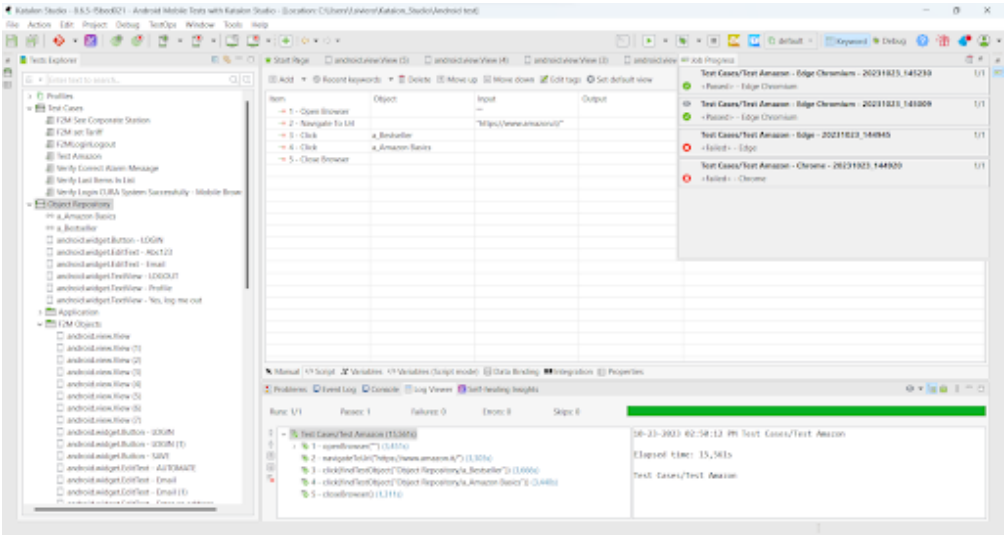

Figure 5.1: Esempio test in esecuzione

Oltre ad eseguire i test case salvati, è anche possibile avviare una nuova registrazione da accodare o sovrascrivere alla precedente.

5.1.5 Debug

Per evitare di eseguire il test case per intero ogni volta che uno step viene modificato, Katalon mette a disposizione una serie di opzioni di debug:

- **Run from here:** questa funzione è disponibile soltanto per alcuni browser (come Chrome, Firefox e Edge Chromium). Data una sessione browser aperta,

è possibile selezionare un qualsiasi test step in Manual View e continuare l'esecuzione del test a partire da quel punto;

- **Enable/Disable steps:** in Manual View è possibile selezionare qualsiasi step e disabilitarlo, in modo che venga saltato durante l'esecuzione;
- **Debug mode:** in Script Mode è possibile impostare dei breakpoint su qualsiasi step e, dopo aver avviato il debug del test attraverso l'omonima opzione, accedere ad una serie di funzioni come suspend/resume, step in, step over e watch di variabili e espressioni;
- **Debug from here:** funzione analoga a "Run from here" per l'esecuzione del test in debug mode

5.1.6 Object Repository

All'interno dell'Object repository, è possibile visualizzare e modificare la strategia di locazione di ciascun oggetto. Per gli elementi web è possibile selezionare tra:

- **Xpath:** viene presentata una serie di possibili xpath tra cui scegliere, i quali possono raggiungere l'elemento in questione tramite i suoi attributi, gli attributi di elementi che lo contengono o la sua posizione nel DOM rispetto ad elementi vicini;
- **Attributes:** un altro approccio basato su xpath in cui si ricerca l'elemento tramite una combinazione di predicati di uguaglianza sui suoi attributi. E' anche possibile scegliere quali attributi utilizzare nel predicato del selettore;
- **CSS:** viene richiesto di inserire un selettore CSS;
- **Image:** viene richiesto il percorso verso l'immagine da utilizzare per la ricerca.

Per gli elementi mobile, in modo simile, è possibile selezionare una strategia di locazione (es. Attributes, Class name, Accessibility ID, XPATH) ed indicare il locatore corrispondente. L'opzione utilizzata di default è "Attributes", che funziona in modo analogo rispetto agli elementi web.

5.1.7 Cucumber

Katalon consente anche di creare ed eseguire dei file .feature, secondo l'approccio BDD. Questi file fanno parte della cartella di progetto "Include/features". Ogni feature è costituita da un nome e un elenco di scenari, a loro volta definiti da un nome e una sequenza di step introdotti da parole chiave Given, When, Then e And. Le definizioni degli step sono contenute nella cartella "Include/scripts/groovy/", e

possono essere suddivise in più package. Si può utilizzare qualsiasi linguaggio di programmazione supportato da Cucumber, come Groovy o Java. Di seguito un esempio di step in linguaggio Groovy:

```
1    @Given("I open the app")
2    def openApp() {
3        Mobile.startApplication('app.apk',true)
4    }
```

In modo analogo, è possibile usare le annotazioni "@Before" e "@After" per definire funzioni da eseguire prima e dopo uno scenario. Oltre agli step, è anche presente una sezione "Examples". Qui vengono definiti i dati da utilizzare negli step parametrici durante l'esecuzione della feature. Essi sono organizzati in una tabella in cui ogni colonna rappresenta un parametro (identificato dal suo nome) e ogni riga rappresenta un set di parametri. Quando l'esecuzione della feature viene lanciata, questa viene ripetuta per ogni riga della tabella.

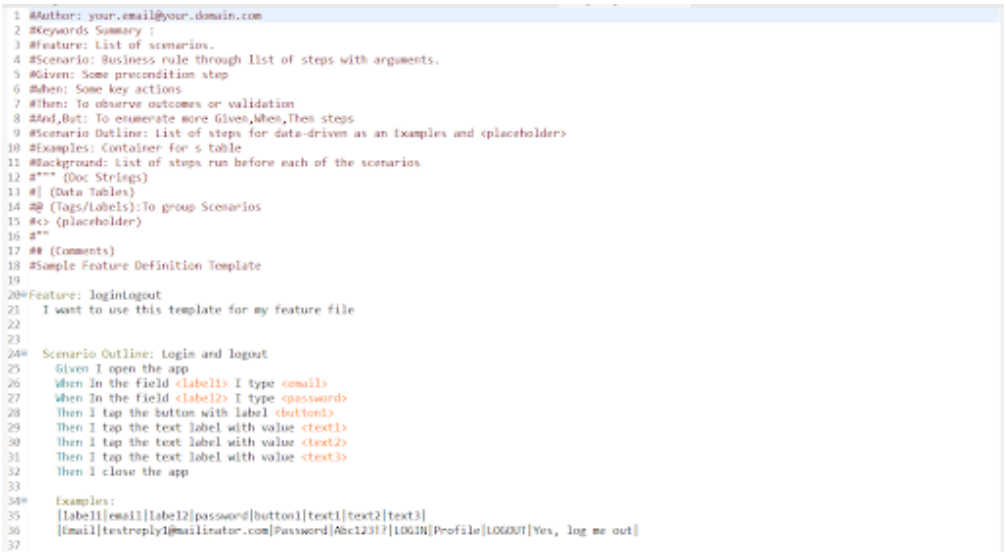

Figure 5.2: Esempio di feature in Katalon

5.1.8 Self-healing

Esclusivamente nell'ambito web, è possibile abilitare l'opzione di self-healing. In questo modo, nei casi in cui non è possibile trovare un elemento utilizzando il suo locatore di default, Katalon sperimenta con dei locatori alternativi associati a quell'oggetto fino a che uno di essi non ha successo. Per configurare la funzione di self-healing, è necessario impostare la priorità delle strategie di locazione utilizzate

(nel caso di Xpath, viene provata ciascuna delle opzioni presenti nel repository per quell'elemento) e definire quali siano le keyword per cui il self-healing non deve essere utilizzato (solitamente, quelle che verificano la presenza o assenza di un dato oggetto).

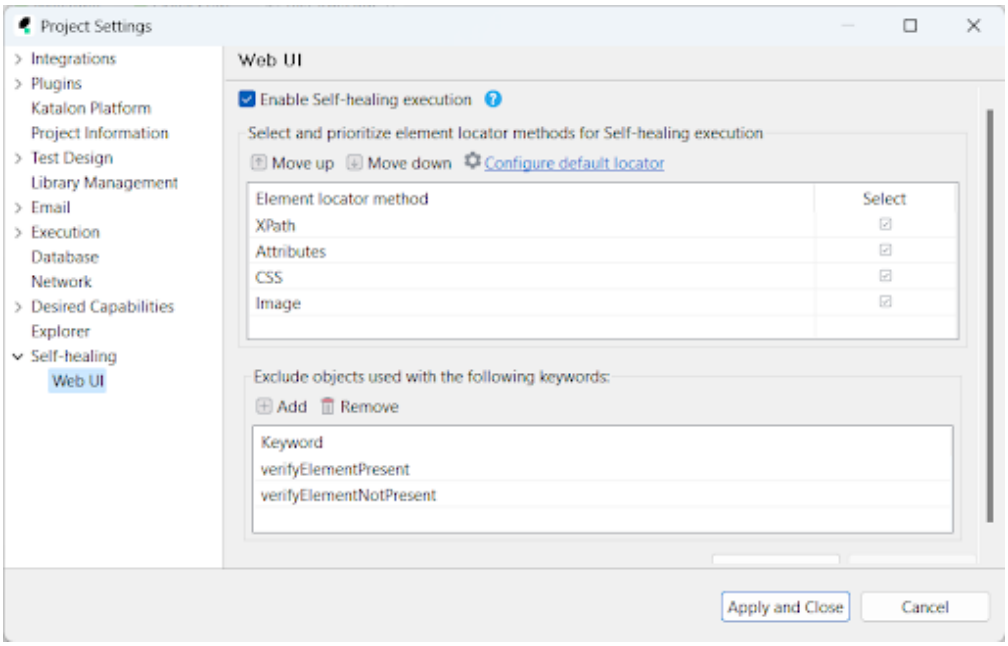

Figure 5.3: Configurazione del self-healing

Durante l'esecuzione, ad ogni fallimento di un locatore, Katalon identifica un sostituto secondo la strategia definita e lo utilizza al posto dell'originale fino al termine dell'esecuzione del test case. Infine, Katalon mostra un report di self-healing che per ogni locatore "rotto" mostra:

- **Test Object ID:** identificatore dell'oggetto non trovato;
- **Broken Locator:** locatore che ha fallito la ricerca dell'elemento;
- **Proposed Locator:** locatore sostitutivo proposto da Katalon;
- **Recovered By:** strategia di locazione usata per identificare il sostituto;
- **Screenshot:** immagine dell'oggetto individuato dal nuovo locatore (in questo modo è possibile verificare che la sostituzione sia valida)

E', quindi, possibile selezionare una o più di queste sostituzioni e renderle persistenti nell'Object Repository.

Esempio di self-healing

Di seguito è presentato un esempio di utilizzo di self-healing all'interno uno scenario di test molto semplice: si accede alla homepage di Amazon, si entra nella sezione "Bestseller" e infine in quella "Amazon Basics":

1. La configurazione di self-healing utilizzata è quella default di Katalon (tutte le strategie abilitate con priorità XPath -> Attributes -> CSS -> Image);
2. Si esegue il test case per la prima volta, in modo da salvare gli elementi con cui è necessario interagire nel repository. Gli xpath definiti da Katalon per identificarli sono
   - //div[@id='nav-xshop']/a[4] (Bestseller)
   - //div[@id='nav-xshop']/a[2] (Amazon Basics)
3. Nella home page di Amazon, si altera l'id del div usato dai due locatori;
4. Eseguendo il test un'altra volta, Katalon fallisce nell'entrare nella sezione Bestseller e, attraverso self-healing, identifica un selettore alternativo (usando uno degli xpath presenti nel repository)

Figure 5.4: Report di self-healing

5.1.9 Gestione degli utenti

Katalon consente di effettuare la gestione di gruppi di utenti. Per fare ciò occorre accedere a Katalon TestOps e definire un'organizzazione, i cui membri hanno la possibilità di lavorare su diversi progetti. Il creatore dell'organizzazione assume di default il ruolo di Owner, ed ha la possibilità di creare nuovi progetti ed invitare nuovi utenti. Questi devono essere in possesso di un account Katalon e, accettando la mail di invito, possono accedere alla loro pagina di TestOps e confermare il loro ingresso nell'organizzazione ed iniziare ad utilizzare la licenza fornita dall'Owner. Gli utenti possono essere suddivisi in più Team, creati dal proprietario dell'organizzazione. Ciascun Team possiede accesso esclusivo a determinati progetti. Il proprietario dell'organizzazione può assegnare ruoli a ciascuno dei suoi utenti. Questi consistono in:

- **Owner:** livello più alto di accesso, Ha la possibilità di gestire in modo completo progetti, team, test e test suite;

- **Administrator:** può creare e gestire team e progetti, aggiungere e rimuovere utenti e assegnare loro dei ruoli. Non ha la possibilità di visualizzare o modificare le informazione personali degli utenti e i dettagli dell'organizzazione;
- **Billing Manager:** si occupa della gestione delle licenze Katalon. Ha la possibilità di gestire le iscrizioni alla piattaforma Katalon e i metodi di pagamento.
- **Member:** livello più basso di accesso, può visualizzare ed eseguire i test ma non può modificare nessuna delle informazioni relative al team o al progetto.
- **User:** ruolo assegnato ad ogni membro dell'organizzazione al momento dell'accettazione dell'invito ad unirsi ad un Team. Consente l'accesso ai progetti assegnati a tale Team.

5.1.10 Considerazioni su Katalon

Dall'utilizzo di Katalon, considerando il paragone con il TAF, è possibile trarre le seguenti conclusioni:

- La possibilità di generare test case in modo immediato sfruttando la feature di recording è molto conveniente: per il non-programmatore è sufficiente agire sul sito/applicazione secondo lo scenario che si intende testare e, automaticamente, viene generato uno script che riproduce le azioni eseguite. Lo script può anche essere modificato in corso d'opera (ad esempio eliminando azioni eseguite erroneamente). Il risultato viene salvato come test case che può essere visualizzato e modificato come script Groovy (da chi possiede le competenze necessarie).
- Oltre a registrare le azioni eseguite, tutti i componenti con cui si interagisce vengono salvati in repository apposito (che può essere organizzato in cartelle) in cui le caratteristiche di ciascun componente possono essere aggiornate (sia manualmente sia tramite la ripetizione di test case). In questo modo è possibile effettuare una gestione degli elementi e dei loro selettori, oltre che dei test case.
- I selettori possono essere generati autonomamente da Katalon, che analizza le caratteristiche dei componenti per costruire xpath efficienti, sollevando in parte lo sviluppatore di test da questo compito.
- L'utilizzo della funzione di Mobile Recording è molto simile ad Appium Inspector: vi sono una device view, la view tree corrente e un insieme di possibili azioni, e selezionando un componente (cliccando direttamente sul

device view o selezionandolo dal view tree) è possibile scegliere l'azione da eseguire su di esso.

- Per utilizzare Appium, Non è necessario mantenere un server in esecuzione in un processo separato. Appium viene avviato automaticamente ad ogni test-run. Non è nemmeno necessario specificare le desired capabilities se il dispositivo di test è direttamente connesso al computer con Katalon in esecuzione.
- Il self-healing è facilmente configurabile e identifica soluzioni alternative in modo chiaro ed efficiente, riducendo di molto lo sforzo di manutenzione necessario per i test su web.
- L'integrazione con Cucumber consente di adottare facilmente un approccio BDD.

Alcuni aspetti negativi da notare sono, però:

- A differenza di Appium Inspector, non è operare sull'applicazione direttamente, bensì è necessario selezionare una delle azioni elencate tra le "Available Actions" e usare la funzione "Capture Objects" per aggiornare la Device View.
- Non è possibile ricorrere a self-healing al di fuori dei test per applicazioni web
- In fase di mobile recording, alcune azioni non sono disponibili e risultano difficilmente replicabili con metodi alternativi. Ad esempio, non è possibile eseguire un pinch per ottenere uno zoom in su uno specifico elemento. Per fare ciò è necessario fornire dei valori specifici per le coordinate del punto di zoom in e l'offset.
- In quanto prodotto su licenza, Katalon manca della flessibilità che caratterizza un sistema sviluppato internamente come il TAF. Non è altrettanto semplice integrarlo con altri sistemi o adattarlo particolari esigenze di test.

In conclusione, Katalon offre molte funzionalità che semplificano la creazione e manutenzione di test ma un'ipotetica transizione da TAF a Katalon richiederebbe un grande sforzo per la conversione di feature, la creazione di un Object Repository e il trasferimento dei profili utente.

5.2 ECU-TEST

All'interno del test configuration file, ad ogni piattaforma utilizzata per il test possono essere associate diverse configurazioni (chiamate "porte"). Ogni job contenuto nei package file è associato ad una specifica porta, di cui ne eredita

le impostazioni. Ad esempio, definendo due o più porte per Appium, è possibile eseguire test che utilizzano molteplici dispositivi mobili contemporaneamente. Questa integrazione, unita alla creazione di test case basata su un'interfaccia grafica, ha il potenziale di semplificare di molto il processo di testing.

5.2.1 Scenario di test

Per valutare la validità di ECU-TEST come strumento di test automation, lo si utilizza per automatizzare il seguente scenario:

1. In un'applicazione Android per la connessione ad autoveicolo, si ricerca sulla mappa una destinazione per la quale si vogliono ottenere indicazioni e la si invia al sistema di infotainment;

2. Si verifica, sulla piattaforma Graylog, che i segnali CAN associati a questo scambio di informazioni siano stati generati correttamente;

3. Infine, sul sistema di infotainment, si verifica la ricezione della destinazione e si avvia la navigazione.

La strumentazione di test prevede, quindi:

- Un dispositivo Android su cui viene eseguita l'applicazione per la ricerca della destinazione;

- Un computer su cui si controlla l'esecuzione del test e si accede a Graylog;

- Il sistema di infotainment del veicolo, connesso ad una test bench che simula la rete CAN;

Quest'ultimo è dotato di un sistema operativo Android, perciò dal punto vista dell'automazione è analogo ad un dispositivo mobile.

Graylog

Graylog è un Log Management System (LMS), una piattaforma in grado di aggregare e organizzare dati provenienti da file di log. L'informazione contenuta in questi file può essere visualizzata attraverso Graylog Search: una pagina in cui è possibile filtrare i dati definendo delle finestre temporali o usando delle apposite query (la cui sintassi è simile a quella di Lucene); il risultato viene visualizzato sia come semplice lista sia come un istogramma raffigurante la distribuzione dei messaggi nel tempo. La ricerca può anche essere salvata o esportata in forma di dashboard.

5.2.2 Configurazione delle interfacce

Ai fini dei test che coinvolgono mobile app e web, i principali tool da configurare sono:

- Appium
- ADB (per sistemi Android)
- Selenium
- Tracetronic Multimedia

Tracetronic Multimedia è un tool fornito da Tracetronic che consente l'elaborazione di video e immagini ottenuti da una camera o memorizzati in un file. Per poterlo utilizzare è necessario configurare il test bench configuration file (fornendo il nome di una porta, il path ad una cartella in cui memorizzare immagini e la lista prioritaria di lingue da utilizzare per OCR) e il test configuration file (indicando la sorgente di video/immagini, sia essa una camera o un file). Multimedia, siccome ECU-TEST è basato su Python, sfrutta la libreria Python OpenCV per fornire funzionalità di image recognition in grado, ad esempio, di identificare testo contenuto in un'immagine o determinare quanto due immagini differiscono tra loro.

Configurazione di Appium

Nel caso di Appium, la configurazione consente di definire ad esempio:

- Ip e porta del server Appium
- Driver da utilizzare
- Piattaforma del dispositivo (es. Android)
- Versione della piattaforma
- Identificatore del dispositivo
- Package/Activity dell'app di interesse (opzionale)

E' anche possibile decidere se attivare il server Appium e/o connettere il server con il dispositivo all'avvio della configurazione o meno e specificare capabilities aggiuntive se necessario. Una volta configurata una porta per Appium, i principali job messi a disposizione per l'interazione con il dispositivo e le sue applicazioni sono:

- **ConnectDevice:** connette il dispositivo all'istanza di Appium in esecuzione

- **DisconnectDevice:** disconnette il dispositivo
- **StartActivity:** avvia una activity dati il suo nome e quello del package che la contiene
- **GetScreenResolution:** restituisce la risoluzione dello schermo
- **SwipeOnScreen:** esegue uno swipe sullo schermo a partire dalle coordinate dei punti di inizio e fine della gesture e dalla durata di quest'ultima
- **PressHardwareButton:** simula la pressione di tasti come back oppure home
- **TouchPosition:** esegue un tocco sullo schermo nel punto indicato dalle coordinate specificate come parametro e mantiene la pressione per la durata indicata

5.2.3 Configurazione Android ADB

Per configurare la porta ADB è sufficiente fornire l'identificatore del dispositivo. Per quanto riguarda i job disponibili, oltre a quelli in comune con la porta Appium come SwipeOnScreen e TouchOnScreen, quelli più degni di nota sono:

- **ExecuteShellCommand:** riceve una stringa corrispondente ad un comando appartenente alla categoria "adb shell" e lo esegue
- **PullFile:** recupera dal dispositivo il file corrispondente al remoteFilePath specificato e lo salva nel localFilePath indicato o in una posizione di default
- **PushFile:** carica nel dispositivo Android alla posizione indicata dal remoteFilePath un file situato nel localFilePath del computer
- **InputText:** inserisce in input il testo specificato, premendo opzionalmente invio

Configurazione di Selenium

Per poter utilizzare Selenium è necessario specificare:

- Nome del browser
- Dimensioni della finestra di visualizzazione del browser
- Path per il file eseguibile del browser
- Path per il WebDriver (non fornito da ECU-TEST, va scaricato separatamente)

La dimensione della finestra può anche essere aggiustata durante l'esecuzione del test con il job SetWindowSize. I principali job per l'interazione con il browser sono:

- **StartBrowser:** avvia il browser secondo le impostazioni del test configuration file
- **StopBrowser:** chiude il browser
- **OpenUrl:** naviga all'url specificato
- **SetWindowSize:** imposta la dimensione della finestra del browser
- **ClickElement:** clicca l'elemento identificato dal locatore passato come parametro
- **ClickPosition:** esegue un click alle coordinate specificate
- **GoBack:** torna alla pagina precedente
- **ScrollElement:** scrolla all'interno dell'elemento definito dal locatore specificato per una distanza specificata (in pixel) lungo l'asse verticale e/o orizzontale.
- **ScrollPage:** scrolla l'intera pagina per la distanza specificata (in pixel) lungo l'asse verticale e/o orizzontale.
- **ScrollToElement:** scrolla la pagina fino a che l'elemento identificato dal locatore passato come parametro diventa visibile
- **WriteText:** scrive la stringa passata come parametro alla posizione corrente del cursore
- **FindElements:** restituisce gli ID degli elementi passati trovati con il locatore specificato

Configurazione di Tracetronic Multimedia

Per usufruire delle funzionalità di image recognition di Tracetronic Multimedia è necessario configurare il test bench configuration file fornendo:

- il nome di una porta
- il path ad una cartella in cui memorizzare immagini;
- la lista prioritaria di lingue da utilizzare per OCR;

E' anche necessario indicare all'interno del Test Configuration File una sorgente di video/immagini, sia essa una camera connessa alla macchina di test o un file.

5.2.4 Creazione del test case

Un test case è rappresentato da una sequenza di azioni fondamentali chiamate "Job". Ognuno di essi può essere fornito direttamente da ECU-TEST (es. Wait) o essere associato ad una delle interfacce definite dal Test Configuration file (es. Selenium). A seconda dell'azione, può essere richiesto l'inserimento di uno o più parametri (in modo simile agli step Cucumber). Nella sezione "Test Case" del package è possibile aggiungere/modificare/eliminare job, cambiarne l'ordine o disabilitarli. Per semplificare la gestione di test lunghi, viene data la possibilità di raggruppare una serie di Job all'interno di un "Block". Ogni blocco può essere modificato allo stesso modo dei singoli Job, ed è anche possibile creare Block annidati. Per inserire un Job all'interno di un block è sufficiente trascinarlo sopra quest'ultimo Il test case, seguendo il flusso definito dallo scenario, è suddiviso in:

- una precondition che avvia l'applicazione mobile;
- un blocco che esegue login, ricerca la destinazione e la invia al sistema veicolare;
- un blocco che accede a Graylog e verifica la presenza dei segnali CAN;
- un blocco che accetta la destinazione sul sistema di infotainment;

Per avviare l'applicazione si utilizza un job che avvia la Activity corrispondente sul dispositivo Android, quindi si tocca lo schermo per chiudere un messaggio di popup. Il blocco che opera sull'applicazione mobile è a sua volta costituito da altri due Block:

1. un blocco che inserisce email e password ed esegue il login;
2. un blocco che cerca la destinazione sulla mappa, la seleziona, la invia al veicolo e verifica l'operazione sia stata completata con successo;

Questo e i successivi step di verifica sono implementati tramite una combinazione delle funzionalità di ADB e Tracetronic Multimedia: usando il job ExcecuteShell-Command si esegue "adb screencap". Questo comando di adb shell produce uno screenshot della schermata corrente e lo memorizza in un path specificato. In questo caso, si salva l'immagine nella scheda SD del dispositivo; quindi con il job PullFile di ADB la si copia dal telefono alla macchina che sta eseguendo il test; infine, con un job ImageRead si analizza lo screenshot e si verifica se questo contiene una determinata stringa o un'immagine di riferimento. Sull'applicazione, ad esempio, dopo aver inviato la destinazione si verifica la comparsa del messaggio "Location was sent'.

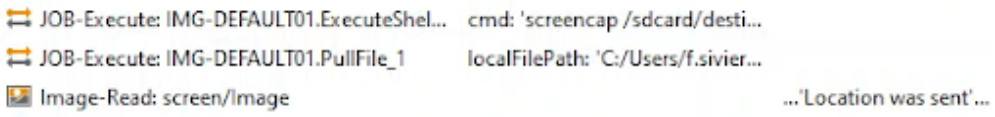

Figure 5.5: Job necessari per la verifica per immagine

Come illustrato nella sezione dedicata alla configurazione, per eseguire un tocco sullo schermo è necessario fornire al job TouchPosition le coordinate xy del punto di contatto. Alcuni dei metodi utilizzabili per ottenerle sono:

- Avviare Appium Inspector, attivare la funzione per inviare un tocco ad un punto dello schermo, portare il cursore nella posizione desiderata e leggere le coordinate visualizzate;

- Abilitare nella sezione "Opzioni sviluppatore" delle impostazioni di Android l'opzione "Posizione puntatore" (in modo da visualizzare le coordinate di tutte le interazioni con il touch screen) e compiere un'esecuzione manuale dello scenario di test;

- Catturare un'immagine della schermata ed individuare i punti di interesse tramite software come Foto o Paint;

Il blocco responsabile dell'interazione con il browser, dopo aver inizializzato il driver, esegue il login su Graylog e verifica nella sezione Search che i segnali CAN siano stati ricevuti. Per fare ciò, si esegue una query che filtra per messaggi di tipo SendGo e si imposta una finestra temporale di visualizzazione sufficientemente stretta da lasciare in vista solamente ciò che è stato generato dal test corrente (in particolare, vengono visualizzati i messaggi inviati fino a 2 minuti prima dell'esecuzione della query). A questo punto, per verificare che il risultato della query sia quello atteso si usa un job di ClickElement con il selettore corrispondente ad una entry della lista di messaggi (se quest'ultima fosse vuota, l'operazione fallirebbe e ciò significherebbe che i messaggi non sono stati ricevuti correttamente).

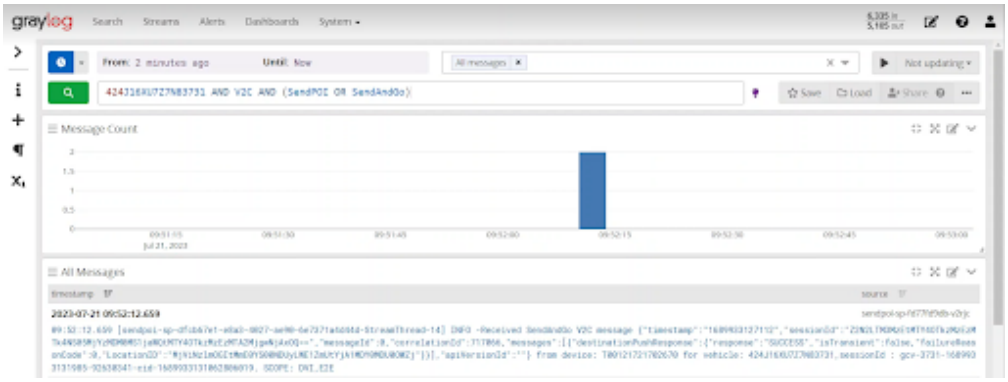

Figure 5.6: Risultato della query su Graylog

Il blocco che interagisce con l'infotainment, dopo aver acquisito uno screenshot e aver verificato la presenza a schermo della stringa "New Destination Received", può seguire due vie alternative:

1. Accettare la destinazione e avviare immediatamente la navigazione;

2. Visualizzare i percorsi alternativi e selezionare uno di essi per la navigazione;

In quest'ultimo caso si verifica anche che la schermata dei percorsi alternativi sia visualizzata correttamente, attraverso l'acquisizione di uno screenshot e la verifica della presenza di una particolare icona. Anche con il sistema di infotainment è necessario determinare le coordinate di ogni tocco eseguito sullo schermo. In questo caso, però, non risulta possibile utilizzare Appium Inspector (probabilmente perché non si tratta di un dispositivo Android convenzionale). Infine, la navigazione sull'infotainment viene interrotta e viene eseguito il logout dall'applicazione Android.

5.2.5 Esecuzione del test

Il package (test) può essere eseguito sia normalmente sia in debug mode (dopo aver fissato degli opportuni breakpoint). Un'altra opzione disponibile è quella per l'esecuzione di test parametrizzati: prima di iniziare, viene mostrato un riepilogo di tutte le variabili d'ambiente definite (con la possibilità di modificarle prima dell'esecuzione del test). Mentre il test è in corso, viene aperta una finestra che mostra l'elenco degli step e quale di essi stia venendo eseguito in tempo reale. Inoltre, per ognuno dei possibili risultati che un test può ritornare (NONE, SUCCESS, INCONCLUSIVE, FAILED, ERROR) viene tenuta traccia del numero di step terminati con quel risultato.

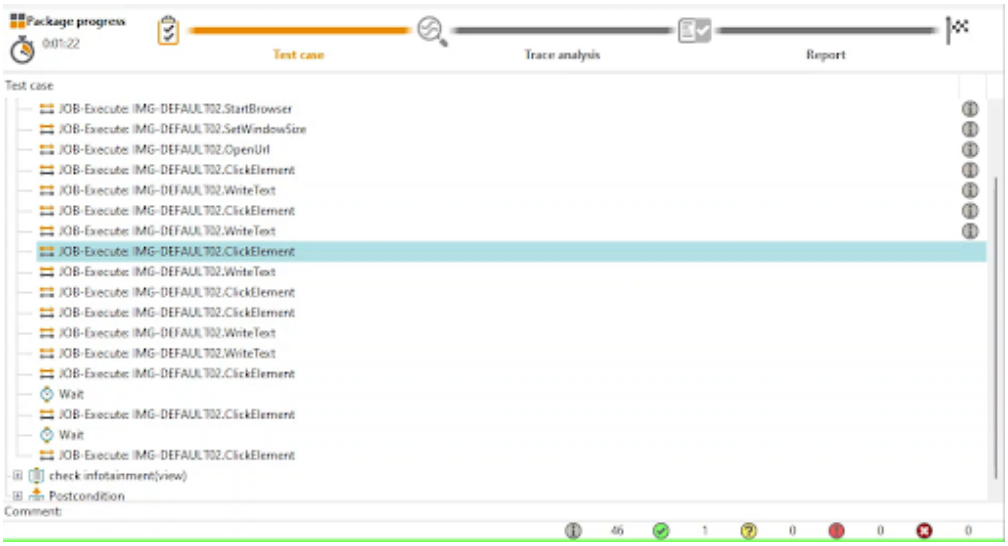

Figure 5.7: Finestra di esecuzione di un package

5.2.6 Test report

Al termine dell'esecuzione, un report viene automaticamente generato e reso disponibile all'interno di un apposito database. Inoltre, ogni volta che un package viene eseguito la finestra "Report Viewer" viene automaticamente aperta. Qui vengono riportate informazioni sul package come ad esempio nome, versione di ECU-TEST, durata dell'esecuzione. Viene, inoltre, mostrato un riepilogo del risultato di ciascuno step (SUCCESS, FAILED, ERROR) e cliccando su uno di essi si mostrano dettagli come tempo di esecuzione e causa di errore (in caso di fallimento). All'interno del report viewer sono anche possibili una serie di azioni come ad esempio:

- **Copy row**: copiare una riga selezionata o tutte le righe del report negli Appunti;
- **Search text:** inserire una stringa di ricerca per saltare direttamente alla sua prima occorrenza all'interno del report;
- **Filter for display:** filtrare gli step visualizzati (ad esempio, è possibile rendere visibili soltanto gli step falliti);
- **Open report folder:** aprire in File Explorer la cartella contenente il test report;
- **Generate test report documents:** generare, a partire dal report, un documento in formato alternativo, come HTML. E' anche possibile aggiungere nuovi

formati da utilizzare all'interno della sezione "Report" del Test Configuration File.

Figure 5.8: Esempio di report

5.2.7 Considerazioni su ECU-TEST

ECU-TEST è una piattaforma commerciale che si pone l'obiettivo di fornire un supporto completo all'esperienza di test, con particolare attenzione al contesto automotive. E' offerto un elevato numero di interfacce configurabili e la creazione di test attraverso la composizione di Job è accessibile anche ad un non-programmatore. La definizione e l'utilizzo di variabili d'ambiente sono immediati e la presenza Job per il controllo del flusso (If-Then-Else) consente di automatizzare scenari più dinamici. Tuttavia, considerando quali sono le esigenze di test di Reply Concept (in particolare l'elevata quantità di test compiuti su browser e dispositivi mobile), emergono una serie di problemi:

- Come menzionato in precedenza, tutte le interazioni rese possibili dalle interfacce di Appium e ADB sono basate su valori di coordinate in pixel e non c'è modo di fare riferimento a nessuno dei componenti della UI visibili a schermo. Ciò è dovuto al fatto che queste integrazioni sono state concepite per essere il più indipendenti possibile dalla piattaforma di utilizzo, pertanto le uniche funzionalità disponibili sono quelle universalmente utilizzabili su qualunque sistema operativo. Per quanto l'interoperabilità possa essere considerata vantaggiosa, la necessità di definire le interazioni con il dispositivo sulla base di coordinate in pixel ha un grave impatto sul tempo di sviluppo dei test e la loro robustezza: oltre al bisogno di verificare manualmente la posizione in cui eseguire ogni singola azione (usando uno dei metodi descritti in precedenza), una qualsiasi variazione dell'interfaccia utente ha la possibilità di rendere il test invalido. Inoltre, è necessario ridefinire le coordinate anche nel caso il test venga eseguito su un dispositivo con uno schermo di dimensione diversa.

- Un altro problema con questo approccio è la difficoltà nella validazione del test stesso: se un'azione viene rivolta verso uno specifico elemento è possibile, ad esempio, osservare che il test sia invalido grazie all'insorgere di eccezioni come NoSuchElement (assumendo che il comportamento dell'applicazione testata coincida con quello atteso). Tuttavia, un'azione basata su coordinate non può fallire fintanto che i valori forniti sono validi. Ciò può portare ad interazioni imprevedibili e non controllate dell'applicazione con conseguenze potenzialmente persistenti.

- Siccome le interazioni basate su coordinate non offrono alcun tipo di feedback, per verificare il test stia procedendo correttamente è necessario acquisire degli screenshot e confrontarli con un'immagine di riferimento (secondo il procedimento mostrato in precedenza). Per quanto questa soluzione sia funzionale, è molto dispendiosa in termini di tempi di sviluppo e di esecuzione.

- Come si è potuto osservare dagli esempi di test mostrati, sono presenti molti blocchi Wait, che non fanno altro che introdurre delle attese statiche. Questi si rendono necessari perché le interazioni basate su coordinate non si basano su nessun timeout per attendere che l'elemento di interesse sia visibile a schermo, quindi il tempo speso dall'applicazione quando si devono ricevere/inviare dati a backend, caricare una nuova schermata o mostrare delle semplici animazioni può essere causa di errori, ed è fondamentale tenerne conto in fase di creazione del test.

La situazione migliora se si considera l'interfaccia di Selenium: vi sono numerosi job che lavorano sulla base di selettori, sono disponibili tutte le principali strategie di selezione e gli step sono facilmente configurabili. Sono anche disponibili job che consentono interazioni come trascinare uno specifico elemento, muovere il cursore su un elemento dato o acquisire uno screenshot della pagina web corrente. Tutte queste operazioni sono controllate da un timeout(...). Nonostante alcune mancanze, come ad esempio l'impossibilità di inviare direttamente del testo ad un elemento specifico e l'assenza di job dedicati ad alcuni elementi particolari come le drop-down list, le azioni possibili consentono di sviluppare test sufficientemente robusti con poca difficoltà.

5.3 CFG

Il CFG (Custom Feature Generator) è un componente del TAF in via di sviluppo. La sua esigenza nasce dal desiderio di alcuni clienti Reply Concept di sfruttare il loro accesso alla piattaforma per creare dei test e modificarli. Per fare ciò, viene presentato un metodo alternativo di creazione di feature file basato sull'uso di un'interfaccia intuitiva e facilmente comprensibile ad un non-programmatore. L'interfaccia di

creazione di feature si basa sulla disposizione, all'interno di un'apposita area di lavoro, di blocchi organizzati in una struttura a flow chart. Ogni blocco rappresenta uno step di esecuzione del test, e corrisponde ad una degli statement Cucumber definiti all'interno della libreria del TAF. All'utente viene presentata una lista di azioni tra cui scegliere, ognuno dei quali può essere selezionato e trascinato nell'area di lavoro per manifestare il blocco corrispondente. Questo può, poi, essere collegato alla struttura preesistente per integrarlo nel flusso di esecuzione. Se lo step a cui il blocco si riferisce contiene dei parametri, viene data all'utente la possibilità di inserirli e il loro valore viene memorizzato all'interno del blocco. A questo strumento è associata una build del TAF dedicata. quando l'utente avvia la creazione del test, un file .feature corrispondente viene creato e aggiunto agli altri test presenti nella build.

5.3.1 Struttura del database

Tutte le informazioni inerenti agli step e alle feature che vengono create sono memorizzate in un insieme di tabelle che si integrano al preesistente database del TAF:

- Statements
- Statement_parameters
- Features

Statements

Questa tabella contiene tutte le informazioni inerenti alle definizioni degli step. Essi sono caratterizzati da:

- **IdStatement:** id univoco autoincrementante
- **Name:** nome dello step, corrispondente a come esso compare nella sua definizione Cucumber. Esso, infatti, rappresenta la base della conversione tra flowchart e file .feature.
- **Description:** breve descrizione dello step e, di fatto, una versione ridotta di ciò che viene espresso dalla definizione Cucumber. La sua funzione principale è quella di migliorare la comprensibilità da parte dell'utente dell'azione svolta dallo step
- **Function:** nome della funzione associata allo step Cucumber, definita all'interno del TAF

- **Platform:** piattaforma per cui lo step viene utilizzato, si usa principalmente per categorizzare gli step. Con "piattaforma" si può intendere sia il tipo di applicazione da testare (Web, Android, iOS) sia una particolare applicazione per cui vengono sviluppati degli step Cucumber specifici. Vi possono anche essere step corrispondenti ad azioni che non sono legate ad un particolare contesto (ad esempio, il wait statico). Tali step sono etichettati come "Generic".

Statement$_{parameters}$

Questa tabella contiene la definizione di ognuno dei parametri eventualmente presenti negli step descritti dalla tabella precedente. Gli attributi che li descrivono sono:

- **IdParam:** id univoco autoincrementante
- **Name:** nome del parametro
- **Type:** tipo del parametro (importante per via della differenza con cui Cucumber tratta stringhe e valori numerici)
- **IdStatement:** id dello statement a cui il parametro appartiene

Features

In questa tabella sono memorizzate informazioni su tutte le feature prodotte da ciascun utente che utilizza il CFG. Esse comprendono:

- **IdFeature:** id univoco autoincrementante
- **Name:** nome della feature, corrispondente anche al nome del file .feature
- **JSON:** rappresentazione del flowchart che descrive la feature, le cui informazioni vengono compilate in un oggetto JSON. Per motivi che verranno illustrati in seguito, però, tale oggetto viene convertito in una rappresentazione numerica prima di essere memorizzato nel db
- **IdUser:** identificatore dell'utente che ha creato la feature
- **IdRole:** identificatore del ruolo dell'utente. Tale informazione può essere utilizzata per introdurre politiche più avanzate di controllo degli accessi

5.3.2 Flowy

Tutte le funzionalità necessarie per la creazione e la visualizzazione dei flowchart provengono da una libreria Javascript detta Flowy. Essa consente di definire handler per eventi come drag, collegamento e cancellazione di un blocco e consente di esportare il contenuto del flow chart in formato JSON tramite la funzione flowy.output(). Il JSON restituito da tale funzione contiene:

- **html:** codice html che descrive la sezione della pagina contenente l'area di lavoro e il flow chart in essa contenuto
- **blockarr:** un array di informazioni posizionali su ciascun blocco, espresse in forma di coordinate xy.
- **blocks:** un array che descrive i blocchi compresi nel flow chart e le informazioni da essi contenute (ad esempio i valori che descrivono uno step parametrizzato). La descrizione di ciascun blocco comprende: il suo id, l'id del blocco padre nel flow chart, un array "data" che contiene i parametri associati al blocco nella forma di oggetti JSON con attributi "name" e "value" e un array "attr" che descrive i tag HTML in cui i parametri sono contenuti tramite oggetti JSON contenenti i loro "id" e "class".

Per utilizzare Flowy all'interno della pagina occorre invocare la funzione flowy() a cui possono essere passati i parametri:

- canvas: elemento HTML che scelto per contenere i flow chart
- onGrab(block): (opzionale) callback invocata ogni volta che un blocco viene trascinato, con il riferimento a tale blocco passato come parametro
- onRelease: (opzionale) callback invocata ogni volta che un blocco viene rilasciato da un drag, sia nel caso esso venga collegato ad un flow chart sia nel caso venga distaccato da quest'ultimo
- onSnap(block, first, parent): (opzionale) callback invocata ogni volta che un blocco viene collegato ad un altro per formare un flow chart. I parametri rappresentano rispettivamente il riferimento al blocco appena aggiunto, un valore booleano pari a true nel caso il blocco sia la radice dello schema e il riferimento al diretto predecessore nella sequenza di blocchi. Questa funzione ha anche un valore di ritorno booleano pari a true se il collegamento viene consentito o pari a false se viene negato. Se questa callback non viene fornita alla creazione di flowy, essa viene sostituita da una funzione che ritorna sempre true.

- onRearrange(block, parent): (opzionale) callback invocata ogni volta che un blocco viene rilasciato da un drag senza che esso possa collegarsi ad altri blocchi. I parametri di questa funzione rappresentano il blocco stesso e l'antenato a cui esso era collegato in precedenza. Se si intende permettere al blocco di staccarsi (ed essere quindi eliminato) la funzione deve ritornare true. Altrimenti, fissando il valore restituito a false, si dispone che il blocco venga ri-agganciato nella sua posizione originaria.
- spacing_x: (opzionale) un valore intero che regola la spaziatura orizzontale tra i blocchi, impostato di default a 20px.
- spacing_y: (opzionale) un valore intero che determina la spaziatura verticale tra i blocchi, impostato di default a 80px.

L'istruzione di inizializzazione di flowy prende, quindi, questa forma:

flowy(document.getElementById("canvas"), onGrab, onRelease, onSnap, onRearrange, spacing_x, spacing_y);

Se le callback per la gestione di snap e rearrange non vengono fornite, flowy adopera delle funzioni di default che staticamente restituiscono true e false rispettivamente (si consente ogni operazione di snap e si blocca ogni rearrange).

5.3.3 My features

Una delle funzionalità fondamentali offerte dal CFG è la possibilità di aprire e modificare feature create attraverso questo strumento. A tale scopo, la prima pagina visualizzata all'apertura del tab CFG nel TAF è "My features".

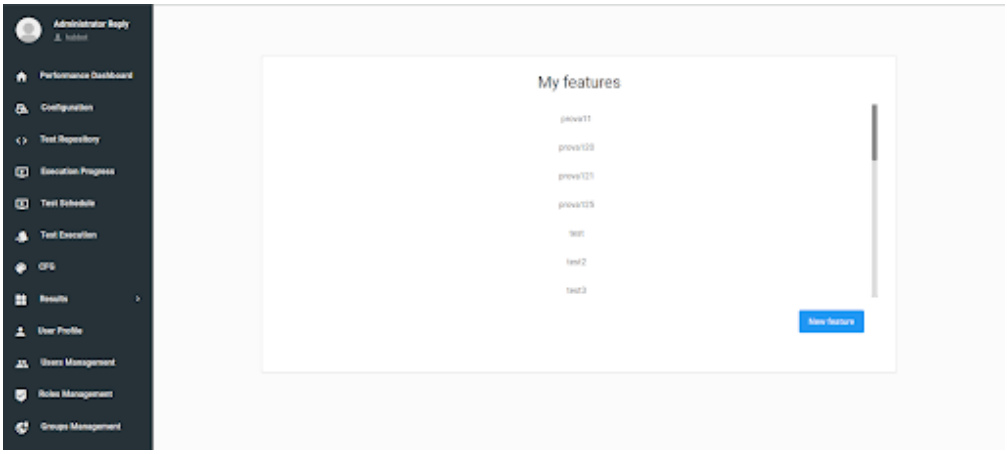

Figure 5.9: Pagina delle feature dell'utente

Essa, all'avvio, recupera dal db l'insieme di tutte le feature create dall'utente corrente e mostra la lista dei loro nomi. A fianco della lista, è presente il bottone che reindirizza direttamente alla pagina di creazione di una nuova feature. Cliccando su uno dei nomi della lista, viene visualizzato un modale che mostra l'insieme di step Cucumber corrispondente al flow chart salvato. Tale lista di statement viene generata a partire dalle informazioni contenute nell'attributo JSON della feature memorizzata a db (previa conversione dell'attributo in un oggetto JSON vero e proprio). Cliccando quindi su "Modify feature" il test viene aperto nell'area di lavoro (il suo id viene passato come parametro URL e usato per recuperare il flow chart corrispondente dal database).

Figure 5.10: Esempio di anteprima della feature

5.3.4 Creazione e modifica di features

La pagina del CFG è suddivisa in 3 sezioni principali:

1. **leftcard:** colonna laterale sinistra, contiene la lista dei blocchi selezionabili e la relativa barra di ricerca

2. **canvas:** sezione centrale della pagina, nonché area in cui il flow chart è contenuto. In alto è anche posta una box di input in cui specificare il nome della feature

3. **rightside:** colonna laterale destra, contiene la feature preview e i bottoni di creazione feature e cancellazione del flow chart

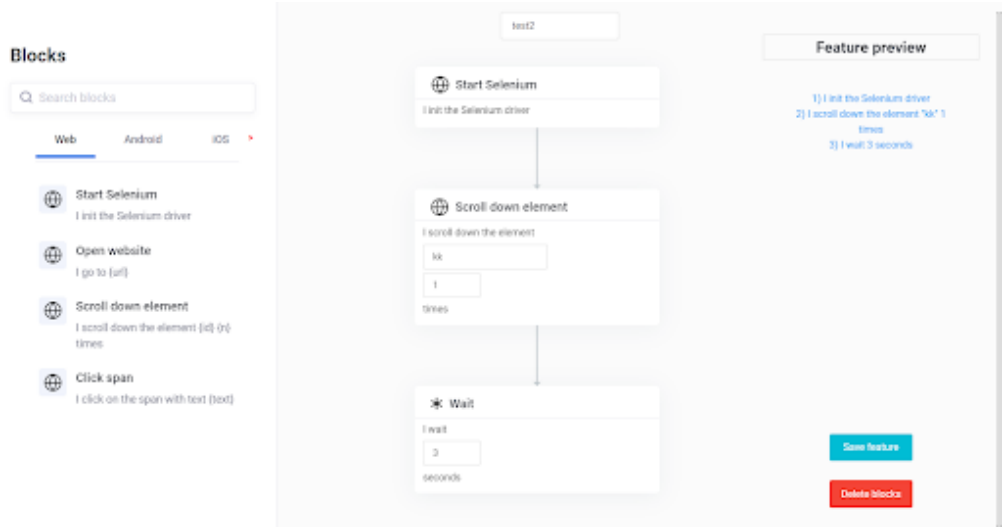

Figure 5.11: Pagina di creazione feature

Leftcard

La sezione sinistra della pagina mostra tutti i blocchi che l'utente può posizionare nel canvas, prelevati dalla tabella tm_statements. La lista è suddivisa in più tab, ognuno dei quali applica un filtro sulla base dell'attributo "platform". Inoltre, viene fornita all'utente una barra di ricerca che mostra gli step per cui la stringa di query sia contenuta negli attributi "name" e "description". Per ogni elemento della lista vengono visualizzati name, description e un'icona della piattaforma di riferimento. All'interno del nome, ogni eventuale parametro viene indicato tramite una coppia di parentesi graffe contenenti una stringa rappresentativa.

canvas

Il canvas è la sezione principale della pagina, in cui è contenuta l'area di lavoro. Il tag corrispondente a questa sezione è ciò che viene utilizzato per inizializzare flowy. Quando un blocco viene trascinato da leftcard e posizionato all'interno del canvas, questo può essere agganciato in coda al flowchart corrente o cancellato (a meno che non si tratti del primo blocco inserito). Per essere agganciato, un blocco deve essere trascinato sufficientemente vicino al lato inferiore dell'ultimo elemento della sequenza da scatenare la comparsa di un indicatore. Rilasciando in quel punto viene avviata la procedura di snap: viene richiamata la callback onSnap e, in caso di risposta pari a true, il nuovo blocco viene automaticamente riposizionato in base alla spaziatura definita e viene disegnata una freccia (arrowblock) per congiungere la vecchia e nuova coda del flowchart. Il caso in cui lo snap viene negato è quello in cui si tenta di creare blocchi paralleli, cercando di agganciare un blocco ad

uno già dotato di successore (siccome gli step all'interno di un file .feature sono prettamente sequenziali). Trascinando il primo blocco della sequenza è possibile riposizionare l'intero flow chart all'interno del canvas. Se, invece, si trascina uno degli altri blocchi, vengono selezionati tutti quelli che seguono. Rilasciandoli viene invocata la funzione di rearrange e, se questa restituisce true e vicino al cursore non è presente alcun punto valido per uno snap, i blocchi vengono eliminati. Questa, infatti, costituisce la procedura standard di flowy per la cancellazione di blocchi singoli o gruppi di blocchi (a patto che questi siano consecutivi e posti in coda alla sequenza).

blocks

I blocchi di flowy sono rappresentati in HTML da div con classe ".create-flowy". Essi possono anche essere arricchiti di parametri attraverso l'inserimento di tag del tipo:

<input type="hidden" name="parameterName" value="parameterValue">

Ognuno di questi parametri corrisponde, all'interno dell'attributo "blocks" del risultato di flowy.output, ad una entry negli array "data" e "attr". Nel caso del CFG, I parametri utilizzati per supportare la creazione di feature sono i seguenti:

- **blockid:** un valore intero gestito autonomamente da flowy che viene incrementato di 1 ogni volta che un blocco viene inserito nel canvas
- **IdStatement:** un valore intero che identifica a livello del db lo step rappresentato dal blocco
- **Name:** una stringa di testo che corrisponde alla feature Cucumber
- **Description:** una stringa di testo che descrive lo step in modo più semplice e conciso rispetto allo statement usato da Cucumber
- **Platform:** una stringa che descrive la piattaforma a cui lo step fa riferimento (es. web, android, ios)

Inoltre, nel caso lo step possegga dei parametri, ognuno di essi è descritto da una serie di input:

- **IdParam:** un valore intero che identifica univocamente il parametro a livello del database
- **IdStatement:** identificatore dello step a cui il parametro appartiene
- **parameter_name:** nome del parametro ottenuto dal db

- **parameter_type:** tipo del parametro (es. string, int), importante per come in Cucumber parametri di tipo diverso abbiano sintassi differente

- **parameter_value:** valore del parametro

Quando un blocco viene prelevato dalla lista laterale e trascinato nel canvas, la posizione di eventuali parametri contenuti nello step viene identificata attraverso la presenza di parentesi graffe, e ciascuna coppia di parentesi viene sostituita da un tag di input in cui l'utente può inserire il valore desiderato. Il tipo di input tag inserito varia a seconda del tipo del parametro: parametri di tipo stringa producono input di tipo "text" mentre input numerici risultano in input di tipo "number". Ogni volta che il valore di un parametro di un blocco viene modificato, l'attributo "parameter_value" corrispondente viene aggiornato.

Creazione del feature file

La funzione fondamentale del CFG è la creazione di feature file sulla base dei flowchart prodotti dall'utente. Ciò prevede non soltanto la costruzione del file .feature, ma anche il suo inserimento in una cartella compressa creata appositamente per feature create tramite il CFG e la ri-compressione di tale cartella. Nel frontend Javascript, vengono eseguite delle verifiche sul flow chart presente nel canvas: si controlla che l'utente abbia inserito un nome per il feature file e che tutti i parametri di tutti i blocchi parametrizzati inseriti abbiano un valore definito. Quindi, viene eseguita una query al db per verificare se una feature con lo stesso nome già esiste. In caso positivo viene mostrato un apposito messaggio, in cui si richiede all'utente se intende proseguire (e sovrascrivere il test preesistente) o annullare l'operazione.

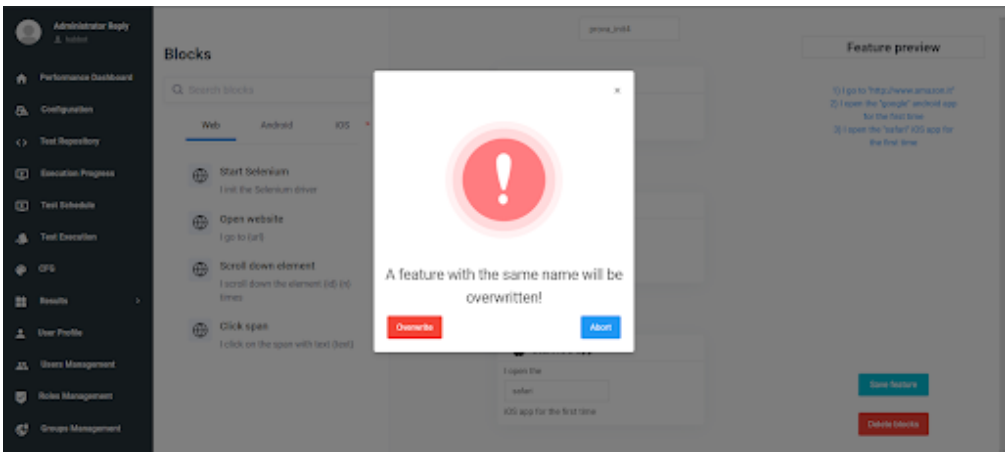

Figure 5.12: Avviso di sovrascrittura

In caso si confermi la sovrascrittura, viene preparato il contenuto del futuro file .feature: il nome fornito dall'utente, oltre a fungere da filename, è usato all'interno del file per denominare feature e scenario (per questo motivo, eventuali spazi presenti nel nome sono rimpiazzati da "_"). A questo punto, si esegue flowy.output() e si itera sul suo "blocks": per ogni blocco contenuto, si preleva la proprietà name per ottenere la "definizione base" dello step; quindi si sostituisce, in modo simile a quanto fatto in precedenza per l'inserimento dei tag di input, ogni coppia di parentesi graffe con il "parameter_value" corrispondente (per come è stato costruito il blocco, le coppie name-value dei parametri seguono l'ordine di apparizione all'interno dello step quindi è sufficiente scorrere l'array "data" linearmente per ottenere la sostituzione corretta); alla stringa risultante viene aggiunta la keyword di Cucumber "Then". Di seguito è presentata la funzione che si occupa di questa procedura.

```
    let json = flowy.output();
let fileContent = [];
let initSteps = [];
let featureName = document.getElementById("featurename").value.
   replaceAll(" ","_");
let scenarioName = featureName;
fileContent.push("Feature: " + featureName + '\n');
fileContent.push("\tScenario: " + scenarioName + '\n');
for(let block of json.blocks){
let blockData = block.data;
let statement = blockData.find(d => d.name === "name").value;
let paramValues = block.data.filter(d => d.name.match(/
   parameter_value_.*/));
let paramTypes = block.data.filter(d => d.name.match(/parameter_type
   /));
for(let i of Array(paramValues.length).keys()){
let val = paramValues[i].value;
switch(paramTypes[i].value){
case "String":
val = '"' + val + '"';
break;
default:
        }
        statement = statement.replace(/\{[^\{\}]*\}/,val);
    }
    fileContent.push("\t\tThen " + statement + '\n');
 }
 let filename = document.getElementById("featurename").value.
   replaceAll(' ','_');
 let steps = "";
 for(let line of fileContent)
    steps += line;
```

Infine viene richiamato il modulo php, eseguendo una richiesta POST ad una API fittizia a cui vengono passati come parametri il nome del file e il contenuto da scrivere in esso, e si attende il risultato. Durante l'attesa viene visualizzato un modale, con cui viene anche comunicato all'utente il successo o fallimento dell'operazione.

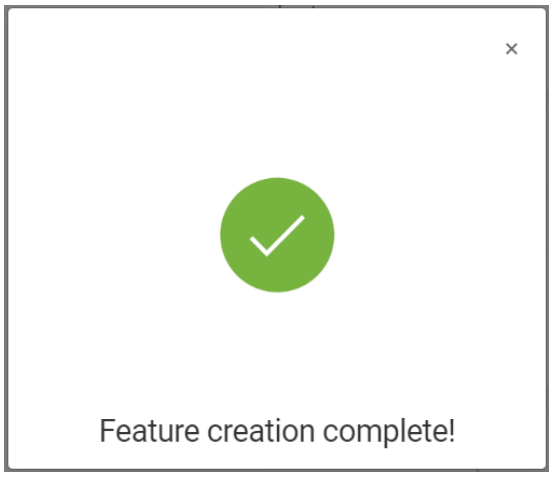

Figure 5.13: Messaggio di successo

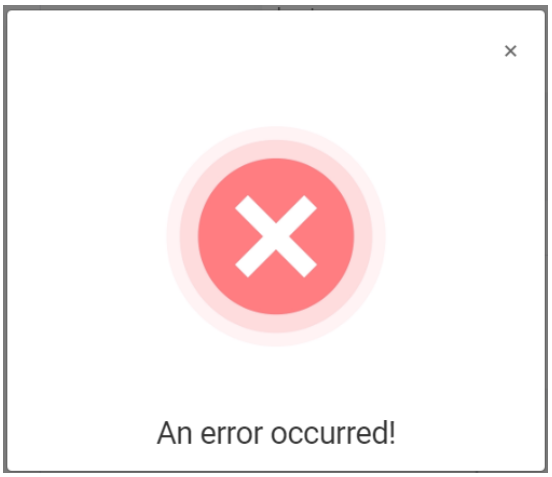

Figure 5.14: Messaggio di errore

Nel backend php, vengono eseguite operazioni sia sul db sia sul file system. Data la necessità di preservare la consistenza tra queste unità di memorizzazione indipendenti, si cerca di imporre una condizione di atomicità: inizialmente, si disabilita la funzionalità di autocommit di mysql e si avvia una transazione. Nel caso in cui emerga un qualsiasi errore tra le operazioni a db e quelle sulla cartella delle build, si esegue un rollback. L'istruzione di commit viene eseguita soltanto al termine di tutta la sequenza di operazioni. A livello del db, si stabilisce una connessione con il database MySQL, quindi si verifica se una feature con lo stesso nome di quella passata come parametro esiste. In caso positivo, è sufficiente aggiornare il record pre-esistente (fondamentalmente, modificare il suo attributo JSON). Altrimenti, viene creata una nuova entry nella tabella. A livello del file system, si decomprime il file ZIP contenente la build dedicata al CFG, si sposta nella cartella appena estratta il contenuto già presente e il file .feature identificato dal nome passato come parametro, si crea una nuova cartella compressa e si rimuove la cartella estratta. Questa operazione è la medesima indipendentemente dal fatto che la feature sia inedita o meno: nel caso di modifica, quello che si ottiene è una sovrascrittura del file .feature precedente. Per evitare conflitti che potrebbero insorgere a causa del tentativo da parte di due utenti diversi di creare feature con

lo stesso nome, ad ogni utente è assegnata ad una cartella. Essa viene creata la prima volta che un qualsiasi utente tenta di salvare una feature.

```
    $zipArchive = new ZipArchive();
    $result = $zipArchive->open("C:/cfg.zip");
    if ($result === TRUE) {
        $zipArchive ->extractTo("C:/cfg");
        $zipArchive ->close();
        $dirPath = 'C:/cfg/features/' . $User->getUsername() . '/';
        if (!file_exists($dirPath)) {
            mkdir($dirPath, 0777, true);
        }
        if(!file_put_contents($dirPath . $parameters->filename . ".
    feature",$parameters->content)){
            $JsonResult["code"] = "Error";
            $JsonResult["message"] = "Failed to unzip/zip directory";
            $DB->rollback();
        }
        $rootPath = realpath('C:/cfg/');
        $zip = new ZipArchive();
            $zip->open('C:/cfg.zip', ZipArchive::CREATE | ZipArchive
    ::OVERWRITE);
        /** @var SplFileInfo[] $files */
        $files = new RecursiveIteratorIterator(
                new RecursiveDirectoryIterator($rootPath),
                RecursiveIteratorIterator::LEAVES_ONLY
            );
            foreach ($files as $name => $file)
            {
                if (!$file->isDir())
                {
                    // Get real and relative path for current file
                    $filePath = $file->getRealPath();
                    $relativePath = substr($filePath, strlen(
    $rootPath) + 1);
                    $zip->addFile($filePath, $relativePath);
                }
            }
            $zip->close();
           function recurseRmdir($dir) {
                $files = array_diff(scandir($dir), array('.','..'));
                foreach ($files as $file) {
                  (is_dir("$dir/$file") && !is_link("$dir/$file")) ?
    recurseRmdir("$dir/$file") : unlink("$dir/$file");
                }
                return rmdir($dir);
              }
            recurseRmdir('C:/cfg/');
```

Rightside e feature preview

La sezione destra della pagina contiene il riepilogo degli step inseriti e le opzioni di salvataggio feature e reset del contenuto del canvas.

I blocchi posizionati nel canvas sono creati da flowy affinché abbiano larghezza ed altezza sufficienti a garantire un'adeguata leggibilità del loro contenuto, e a tale scopo contribuisce anche la lunghezza delle frecce che li connettono. Tuttavia, ciò comporta che 3-4 blocchi collegati siano sufficienti e riempire lo spazio disponibile nella finestra quando un singolo test può richiedere decine di blocchi per essere implementato. Anche dando all'utente la possibilità di scrollare il canvas per navigare il flow chart, ciò non soddisferebbe le necessità di navigare il flow chart in modo efficiente e di poter mantenere una visione d'insieme della feature in via di scrittura. A tale scopo, sul lato destro del canvas viene mostrata la feature preview: una lista scorrevole contenente, per ciascun blocco, una stringa corrispondente a come esso verrà tradotto in step di Cucumber nel file .feature. Inoltre, ciascuna delle stringhe agisce come link verso il blocco corrispondente, permettendo all'utente di muoversi velocemente da un estremo all'altro del flow chart. Per realizzare la feature preview, si sono sfruttate principalmente la funzione flowy.output() per ottenere tutte le informazioni necessarie dal flow chart e la proprietà "blockid" gestita da flowy: siccome blockid assume un valore univoco per ogni data configurazione del flow chart, è possibile assegnare all'attributo "id" di ciascun blocco un valore legato a "blockid" e costruire dei link che collegano ad esso. Per ottenere il testo del link, si utilizza flowy.output() in modo analogo a quanto fatto per la creazione del contenuto del feature file. La feature preview deve essere aggiornata ogni volta che il flow chart viene modificato. Ciò viene fatto attraverso una apposita funzione di aggiornamento (updatePreview()). Cambiamenti nello schema a blocchi possono consistere nella modifica di parametri o nella creazione/eliminazione di blocchi. Per gestire il primo caso, siccome si tratta di normali eventi html, è sufficiente usare la funzione updatePreview() come handler di onChange() o onInput(). Per gestire il secondo caso occorre modificare la funzione drag di flowy. In particolare, è stato aggiunto al costruttore di flowy il parametro after_drag. Questo corrisponde ad una funzione da eseguire al termine di ogni operazione di drag.

5.3.5 Miglioramenti futuri

Nel suo stato corrente il CFG è in grado di generare feature in modo affidabile. A breve termine si può considerare la possibilità di completare l'integrazione con il TAF, ad esempio facendo in modo che oltre a creare il file .feature il test corrispondente venga aggiunto alla sezione di Test Repository, in modo da poterlo

eseguire attraverso la pagina di Test Execution. Inoltre, sono molti i miglioramenti possibili all'esperienza utente (ciò è dovuto alle limitazioni intrinseche di flowy). Ad esempio, al momento è impossibile scambiare di posizione dei blocchi già posizionati o eliminare un singolo step che non si trovi in coda al flow chart. Ragionando a lungo termine, si potrebbe associare il CFG ad una funzionalità di record e playback simili a quelle di Katalon Studio. Attualmente, ogni feature è visibile e modificabile esclusivamente dall'utente che l'ha creata. Per gestione degli utenti più avanzata, si possono definire diritti di accesso alle diverse feature in base ai ruoli e gruppi presenti nel TAF. Ad esempio, si possono definire feature esclusive per un determinato gruppo, o dare ad utenti con ruolo di ADMIN l'accesso a test creati da utenti di ruolo inferiore. Un'altra possibile miglioria è l'introduzione di blocchi ad alto livello: invece di avere una corrispondenza 1:1 tra step Cucumber e block del CFG, si potrebbero definire blocchi associati ad azioni più complesse come eseguire login (dati username e password). Un'ulteriore prospettiva futura è l'integrazione con strumenti di AI. Le possibilità comprendono ad esempio il training di modelli per la creazione automatica di feature dato un insieme di requisiti e l'introduzione di funzionalità di self-healing.

Chapter 6

Conclusioni

Come affermato nell'introduzione, l'obiettivo è quello di effettuare una procedura di testing efficace che minimizzi il time to market. Le aree su cui è possibile intervenire sono molteplici: si può ridurre il tempo speso in fase di regression testing (attraverso funzionalità come il self-healing); si può semplificare la scrittura di test case in modo da renderla più rapida ed accessibile (sia attraverso la generazione automatica di script tipica del record-and-playback sia attraverso l'utilizzo di interfacce utente intuitive che facilitano l'utilizzo delle funzioni offerte da un Test Automation Framework). Potenzialmente, si potrebbe pensare di automatizzare lo stesso test design (attraverso la funzione di spidering citata in precedenza). Ogni soluzione che si considera adottare deve essere valutata in base a quanto questa sia compatibile con le esigenze di testing correnti e quale sia l'entità del suo impatto sui processi di testing già definiti (in termini di costo di transizione). Il framework di Reply Concept è una piattaforma versatile, nata dal desiderio di attuare un processo di testing specializzato e trasparente. Ognuna delle alternative presentate offre la possibilità di produrre test in modo più rapido ed accessibile, ed eventualmente ridurre lo sforzo dedicato alla gestione. Le soluzioni basate su strumenti commerciali sono quelle che consentono una più immediata introduzione di nuove funzionalità (come ad esempio self-healing e record-and-playback). Tuttavia, esse rappresentano anche un grande distacco rispetto alla piattaforma attuale: un'eventuale transizione di test case, librerie di funzioni e gestione degli utenti richiederebbe uno sforzo tale da rendere potenzialmente irrilevanti le nuove funzionalità introdotte, senza contare il rallentamento che ciò porterebbe alla produzione di test corrente. Inoltre, affidarsi ad una piattaforma esterno significa rinunciare alla flessibilità e capacità di adattamento offerta dal TAF. Alcune delle funzionalità cardine del framework corrente potrebbero non trovare un loro corrispettivo, o essere disponibili in una forma che non viene incontro alle esigenze di test di Reply Concept. La creazione di nuovi strumenti come il CFG è l'alternativa più immediata per quanto riguarda

l'integrazione con la piattaforma corrente, e come per il TAF si possono implementare ed adattare le funzionalità necessarie in modo da soddisfare al meglio le esigenze di testing. Tuttavia, introdurre funzionalità nuove e rivoluzionarie richiede un grande sforzo di ricerca: occorre acquisire una comprensione profonda delle tecnologie di interesse e determinare come queste possono essere integrate utilizzando strumenti già presenti nella piattaforma corrente o del tutto nuovi. Questo tipo di approccio, però, risulta più sostenibile poiché maggiormente compatibile con una conduzione in parallelo rispetto ai processi di testing preesistenti.

Lavori futuri

Considerando la situazione attuale e le alternative presentate, alcune prospettive future sono:

- Esplorare ulteriormente la piattaforma di Katalon, nel modo in cui questa può offrire funzioni equivalenti a quelle del TAF attuale e metterne a disposizione delle altre. In alternativa, nel caso in cui il cambio di framework non risulti praticabile, è possibile ricercare dei metodi alternativi per integrare nel TAF strumenti potenzialmente molto utili come record-and-playback e self-healing, facendo riferimento a Katalon per il modo in cui vengono utilizzati;
- Identificare situazioni di utilizzo di ECU-TEST in cui non vengono evidenziate alcune delle sue maggiori lacune (in particolare, scenari di test che non coinvolgono applicazioni mobile), e si risaltano invece la sua accessibilità e la sua specificità rispetto al contesto automotive;
- Sviluppare ulteriormente il CFG, rifinendo le funzionalità già presenti (ad esempio, rendendo la costruzione dei flow chart più semplice e intuitiva), migliorando il livello di integrazione con il TAF (lavorando in particolare sulla gestione degli utenti e il controllo degli accessi) e valutando la possibilità di introdurre nuove macro-funzioni come record-and-playback, self-healing e generazione automatica di test case.

Bibliography

[1] Thomas Zurawka and Joerg Schaeuffele. *Automotive Software Engineering*. 2nd ed. SAE International, 2016 (cit. on pp. 4, 9).

[2] Michael Meisinger Manfred Broy Ingolf H. Krüger. *Automotive Software – Connected Services in Mobile Networks. First Automotive Software Workshop, ASWSD 2004 San Diego, CA, USA, January 10-12, 2004 Revised Selected Papers*. Springer, 2006 (cit. on pp. 4, 8).

[3] Boby Jose. *Test Automation. A manager's guide*. 1st ed. BCS Learning & Development Limited, 2021 (cit. on pp. 16, 19, 22, 23).

[4] *Android ADB*. URL: `https://developer.android.com/tools/adb?hl=it` (cit. on p. 23).

[5] David Burns Simon Stewart, ed. *WebDriver W3C Editor's Draft*. URL: `https://w3c.github.io/webdriver/` (cit. on p. 26).

[6] *Appium Documentation*. URL: `https://appium.io/docs/en/2.1/` (cit. on p. 29).

[7] *UiAutomator2*. URL: `https://github.com/appium/appium-uiautomator2-driver` (cit. on pp. 31, 32).

[8] *WebDriverAgent*. URL: `https://github.com/facebookarchive/WebDriverAgent` (cit. on p. 32).

[9] *NGA: how Self Healing Automation works?* URL: `https://www.nextgenerationautomation.com/post/how-self-healing-automation-works` (cit. on p. 34).

[10] Joe Colantonio. *AI Test: How It's Changing Test Automation (+6 Examples)*. 2018. URL: `https://testguild.com/how-ai-is-changing-test-automation` (cit. on p. 35).

[11] *Katalon Documentation*. URL: `https://docs.katalon.com/docs` (cit. on p. 35).

[12] Federico Meloni. «Competitività e risorse: Il caso Concept Reply». PhD thesis (cit. on p. 38).

[13] Robert Oshana & Mark Krealing. *Software Engineering for Embedded Systems*. 1st ed. Elsevier Science & Technology, 2013.

[14] Richard E. Fairley. *Systems Engineering of Software-Enabled Systems*. 1st ed. IEEE Press Series. John Wiley & Sons, Incorporated, 2019.

[15] *Selenium Documentation*. URL: https://www.selenium.dev/documentation/webdriver/.

[16] Mark Collin. *Mastering Selenium WebDriver 3. 0. Boost the Performance and Reliability of Your Automated Checks by Mastering Selenium WebDriver, 2nd Edition*. 2nd ed. Packt Publishing, Limited, 2018.

[17] Hugo Peres. *Automating Software Tests Using Selenium*. Scribl, 2018.

[18] *Appium Inspector*. URL: https://github.com/appium/appium-inspector.

[19] Rex Black, James Davenport, Joanna Olszewska, Jeremias Rößler, Adam Leon Smith, and Jonathon Wright. *Artificial Intelligence and Software Testing. Building systems you can trust*. BCS Learning & Development Limited, 2022.

[20] *Cucumber Documentation*. URL: https://cucumber.io/docs/cucumber/.

[21] *Healinium documentation*. URL: https://healenium.io/docs/how_healenium_works.

[22] *ECU-TEST*. URL: https://www.tracetronic.com/products/ecu-test/.

[23] *Graylog Documentation*. URL: https://go2docs.graylog.org/5-1/what_is_graylog/what_is_graylog.htm.