UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

SCUOLA POLITECNICA E DELLE SCIENZE DI BASE

DIPARTIMENTO DI INGEGNERIA ELETTRICA

E TECNOLOGIE DELL'INFORMAZIONE

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA BIOMEDICA

(CLASSE DELLE LAUREE MAGISTRALI IN INGEGNERIA BIOMEDICA LM-21)

TESI DI LAUREA

TESTING DI APPLICAZIONI ANDROID CON TECNICHE DI CAPTURE & REPLAY

RELATORE

CH.MO PROF. PORFIRIO TRAMONTANA

DIPARTIMENTO DI INGEGNERIA INFORMATICA

CANDIDATO

CLAUDIA ORLANDO

M54/69

ANNO ACCADEMICO 2020/2021

INDICE

1. Introduzione

Le piattaforme mobili stanno diventando sempre più diffuse e, conseguentemente, le applicazioni mobili (o semplicemente le app) che funzionano su tali piattaforme. Oggigiorno, utilizziamo app per molte delle nostre attività quotidiane: dagli acquisti, alle attività bancarie, ma anche i social network e persino per i viaggi. Come per le altre applicazioni software, le app devono essere testate per assicurarsi che si comportino correttamente ad input e condizioni diverse. Ciò è particolarmente importante al giorno d'oggi, dato il numero elevato di aziende che rendono disponibili le proprie app agli utenti, poiché i malfunzionamenti di un'app possono comportare una perdita di reputazione e, di conseguenza, anche di clienti, per l'azienda stessa che fornisce l'app. Da qui, quindi, il motivo per il quale le aziende spendono considerevoli quantità di denaro e di risorse per la cosiddetta attività di Quality Assurance (QA) e, in particolar modo, per l'attività di testing [2]. Nel caso delle app Android, il panorama è ulteriormente complicato a causa della frammentazione dell'ecosistema Android, che include innumerevoli dispositivi disponibili in tutte le forme e dimensioni e che possono eseguire diverse versioni del sistema Android. Ottenere la sicurezza che un'app funzioni correttamente su tutta la gamma di dispositivi Android e la versione del sistema operativo corrispondente è particolarmente impegnativo ed oneroso [15]. Tuttavia, il testing delle app mobili presenta alcuni problemi specifici. Poiché la maggior parte dell'interazioni con un'app avviene mediante la sua Interfaccia Utente (Graphical User Interface – GUI), è fondamentale eseguire anche un test estensivo a livello GUI, ad esempio considerando un'applicazione sotto test (Application Under Test – AUT) inviando eventi di input alla sua interfaccia. Inoltre, come già detto, la frammentazione dei sistemi operativi e dei dispositivi mobili causa la necessità di ripetere i medesimi test su un numero elevato di differenti dispositivi ed ambienti di esecuzione. Esistono tre principali approcci che consentono di testare le GUI delle applicazioni mobili. Il primo è il tradizionale test-case basato sul testing del software nel quale i test case sono precedentemente progettati, poi, implementati ed infine eseguiti. Il secondo approccio è basato, invece, sui tool di generazione automatica di input (Automated Input Generation – AIG), con l'obiettivo di stressare le GUI di un'app generando automaticamente sequenze di eventi input. Una terza alternativa è rappresentata dal testing esplorativo (Exploratory Testing – ET). Esso è un tipo di test manuale che è ancora ampiamente apprezzato ed utilizzato nell'industria del software. Si tratta di un approccio basato sull'esperienza creativa in cui la progettazione, l'esecuzione e l'apprendimento dei test sono attività parallele ed i risultati dei test eseguiti vengono immediatamente

Commentato [PT1]: Di solito si utilizza il paragrafo giustificato invece che allineato a sinistra, ma ovviamente sono scelte personali

applicati per progettare ulteriori test. Il testing esplorativo può essere supportato dai tool di Capture e Replay (Cattura e Replicazione - C&R). Essi, senza richiedere competenze di programmazione e di testing avanzate, generano automaticamente degli script di test da sequenze di interazione dell'utente con AUT (come inserimenti da tastiera o movimenti del mouse), acquisendo, in questo modo, scenari di utilizzo reali. A partire da questo script, il tool riproduce sul programma, o successive versioni del medesimo, le operazioni registrate senza richiedere l'intervento attivo di un utente. La nostra ricerca sarà, quindi, orientata allo studio dei tool di Capture e Replay (C&R), attualmente in commercio, esaminando, per ognuno di essi, le caratteristiche ed il loro scenario di utilizzo in modo da stabilire quale possa essere il tool di nostro utilizzo partendo dalle condizioni in cui abbiamo a disposizione il solo APK.

2. Analisi degli strumenti disponibili

In questo capitolo, andremo ad effettuare un'analisi sui tool disponibili di Capture & Replay e, successivamente, a valle di questa analisi, valuteremo il tool di C&R che confà ai nostri criteri.

2.1 Sikuli

Commentato [PT2]: Ti conveniva utilizzare negli stile di word lo stile Titolo1, Titolo2, etc. per capitoli e paragrafi per poi farti generare l'indice automaticamente da word, che lo avrebbe anche aggiornato su richiesta in caso di cambiamenti

Sikuli è un tool basato su un approccio visivo finalizzato alla ricerca e all'automazione di interfacce utente (GUI) mediante l'utilizzo di screenshot [8]. Inoltre, il tool consente agli utenti di fare uno screenshot di un elemento GUI (come un pulsante della barra degli strumenti, un'icona o una finestra di dialogo) ed interrogare il sistema usando lo screenshot invece del nome/ID dell'elemento. Sikuli inoltre fornisce un API di scripting visuale per automatizzare le interazioni con le GUI usando pattern di schermate per dirigere eventi del mouse e della tastiera. Nella lingua indiana, *Sikuli* significa "Occhio di Dio", simbolo del potere di vedere e di capire cose sconosciute. Sikuli consente di creare un riferimento visivo agli elementi GUI. L'utente può disegnare un rettangolo intorno ad un elemento GUI di cui vuole conoscere il database e fare un screenshot come una query. Allo stesso modo, per automatizzare le interazioni con un elemento GUI, il programmatore può inserire lo screenshot dell'elemento nello script e specificare quali azioni della tastiera o del mouse invocare, quindi, questo elemento viene visualizzato sullo schermo. Rispetto alle alternative non visive, gli screenshot rappresentano un modo intuitivo per specificare una varietà di elementi GUI. Inoltre, dal momento che è sempre possibile fare uno screenshot di un elemento GUI, gli screenshot sono universalmente accessibili per tutte le applicazioni su tutte le piattaforme GUI. A differenza di altri meccanismi, come descrizione comandi o tasti di scelta rapida (F1) che potrebbero o meno essere implementati nell'applicazione. Infatti, gli approcci attuali richiedono agli utenti di inserire parole chiave per gli elementi della GUI al fine di trovare informazioni su di essi. Tuttavia, le parole chiave adatte potrebbero non essere immediatamente ovvie. Da qui, quindi, l'utilità di utilizzare gli screenshot dell'elemento come una query. Gli elementi GUI possono essere rappresentati più facilmente dagli screenshot. *Sikuli Search* è un sistema che consente agli utenti di cercare una vasta raccolta di documentazione online sugli elementi GUI usando gli screenshot. *Sikuli Search* consta di tre componenti: un motore di ricerca di screenshot, un'interfaccia utente per interrogare il motore di ricerca ed un'interfaccia utente per aggiungere screenshot con annotazioni personalizzate all'indice. Il sistema indicizza schermate estratte da una vasta gamma di risorse come tutorial online e documentazione ufficiale. Il sistema rappresenta ogni screenshot usando tre differenti tipi di funzionalità. Gli screenshot possono essere indicizzati in base al testo circostante, alle caratteristiche visive ed al test incorporato (via OCR). Secondo il primo approccio, usiamo il testo che circonda lo

screenshot nel documento di origine, che è un approccio tipico adottato dagli attuali motori di ricerca di immagini di base di parole chiave. Nel secondo approccio, lo screenshot viene indicizzato in base alle caratteristiche visive. Recenti progressi nella visione artificiale hanno mostrato l'efficacia della rappresentazione di un'immagine come un insieme di parole visive. Una parola visiva è un vettore di valori calcolati per descrivere le proprietà visive di una piccola patch di un'immagine. Le patch sono generalmente campionate da posizioni salienti dell'immagine come angoli che possono essere rilevati in modo affidabile nonostante le variazioni di scala, traslazione, luminosità e rotazione. Il terzo approccio, infine, indicizza gli screenshot in base al testo incorporato. Dal momento che molto spesso gli elementi GUI contengono un testo, possiamo indicizzare gli screenshot di questi in base al testo incorporato estratto in base alla tenica del riconoscimento ottimo dei caratteri (Optical Character Recognition – OCR). *Sikuli Search* consente, inoltre, di selezionare una regione di interesse sullo schermo, inviare l'immagine nella regione come query al motore di ricerca e osservare i risultati della ricerca. Per specificare la regione di interesse, bisogna premere un tasto di scelta rapida in modo da passare alla modalità di ricerca Sikuli. A quel punto, bisogna trascinare una sorta di riquadro attorno alla regione di interesse. Gli utenti, inoltre, non hanno la necessità di adattare perfettamente il rettangolo attorno all'elemento GUI desiderato. Una volta che il rettangolo è stato disegnato, accanto ad esso, compare un pulsante di ricerca che invia l'immagina nel rettangolo come una query al motore di ricerca e, successivamente, apre il browser per visualizzare i risultati. *Sikuli Script* rappresenta un approccio visivo all'automazione dell'interfaccia utente mediante l'ausilio di screenshot. *Sikuli Script* ha diverse componenti. La funzione **find()** individua un particolare elemento GUI con il quale interagire. Tale funzione prende un modello visivo che specifica l'aspetto dell'elemento, cerca l'intero schermo o parte di esso, ed infine restituisce le regioni corrispondenti a questo modello. In alternativa, restituisce *false,* laddove, invece, non fosse possibile trovare tale regione. Ad esempio 

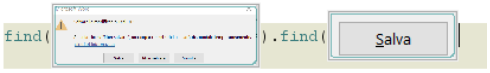
, restituisce regioni contenenti icone del documento Word. La classe del modello è un'astrazione per i modelli visivi. Un oggetto pattern può essere creato da un'immagine. Un oggetto pattern basato su un'immagine ha quattro metodi per regolare quanto devono essere generali o specifiche le corrispondenze desiderate:

- **exact():** richiedono che le corrispondenze siano identiche al dato modello di ricerca pixel per pixel
- **similar:** consentire corrispondenze leggermente diverse dal modello specificato. Una soglia di somiglianza tra 0 ed 1 specifica quanto devono essere simili le regioni corrispondenti.
- **anyColor():** consentire corrispondenze con colori differenti rispetto al modello indicato.
- **anySize():** consentire corrispondenze di dimensioni differenti rispetto al modello indicato.

Ciascun metodo produce un nuovo modello, quindi possono essere concatenati. Per esempio:

Corrisponde all'area dello schermo che è all'80% simile a  di qualsiasi dimensione e composizione di colore. La classe Region fornisce un'astrazione per la regione dello schermo restituita dalla funzione **find()** corrispondente ad un determinato modello visivo. I suoi attributi sono le coordinate x ed y, l'altezza e la larghezza. Generalmente, un oggetto Region rappresenta la corrispondenza superiore, per esempio, 

, trova la regione più simile a  e la assegna alla variabile *r*. Un oggetto Region, se usato insieme ad un'istruzione iterativa, rappresenta una matrice di corrispondenze. Per esempio, per *r* in 

, itera attraverso una matrice di regioni corrispondenti ed il programmatore può specificare quali operazioni eseguire su ciascuna regione rappresentata da *r*. Un altro uso dell'oggetto Region consiste nel vincolare la ricerca ad una particolare regione anzicchè all'intero schermo. Per esempio, 

, limita lo spazio di ricerca del secondo **find()** per il pulsante Salva solo alla regione occupata dalla finestra di dialogo restituita dal primo **find()**. I comandi di azione specificano quali eventi di parole chiave e/o mouse devono essere emessi al centro di una regione trovata dall'oggetto **find()**. Il set di comandi attualmente supportati sono :

- **click**(Region)**, doubleClick**(Region) : questi due comandi inviano eventi di click del mouse al centro di una regione target.
- **dragDrop**(Region target, Region destination) : questo comando, invece, trascina l'elemento al centro di una regione target e lo rilascia al centro di una regione di destinazione. Per esempio, 

, trascina l'icona Word e la rilascia nel Cestino.
- **Type**(Region target, String testo): questo comando inserisce un dato testo in una regione target inviando i tasti al suo centro.

Inoltre, Sikuli prevede la presenza di un editor che consente agli utenti la scrittura di visual script. Per fare uno script di un elemento GUI da aggiungere ad uno script, l'utente può cliccare sul

pulsante della fotocamera nella barra degli strumenti per accendere alla modalità di acquisizione dello schermo. L'editor si nasconde automaticamente per rivelare il desktop sottostante e, conseguentemente, l'utente può disegnare un rettangolo attorno ad un elemento GUI per catturare il suo screenshot. L'immagine acquisita può essere incorporata in qualsiasi istruzione e visualizzata come immagine incorporata. L'editor fornisce inoltre la possibilità di completamento del codice. Quando l'utente digita un comando, l'editor visualizza automaticamente il modello di comando corrispondente per ricordare all'utente quali argomenti aggiungere. Ad esempio, quando l'utente digita il comando **find**, l'editor espande il comando in  . L'utente può cliccare sul pulsante della fotocamera per acquisire uno screenshot come argomento per questa istruzione **find()**. In alternativa, l'utente può caricare un file immagine esistente dal disco rigido o digitare il nome del file o l'url di un'immagine. Successivamente, l'editor caricherà quest'ultima e lo visualizza come anteprima. L'editor inoltre consente all'utente di specificare una regione arbitraria dello schermo per limitare la ricerca a quella regione. Infine, l'utente preme il pulsante di esecuzione. L'editor verrà nascosto e lo script verrà eseguito. L'editor può anche visualizzare l'anteprima di come un modello corrisponde al desktop corrente con parametri differenti quali una soglia di somiglianza ed un numero massimo di corrispondenze. In modo tale che questi possano essere regolati per includere solo le regioni desiderate.

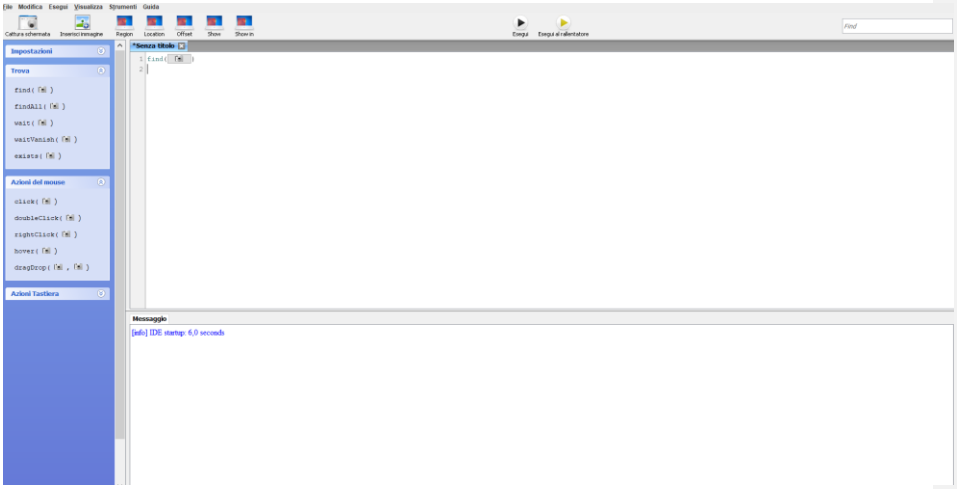


Fig2.1 Editor in Sikuli per la scrittura degli script in Python.

2.2 RANDR

Esso è un tool di Capture e Replay che consente la registrazione e la riproduzione di più fonti di input, evitando le restrizioni che minano la praticità [9]. Il tool si aggancia ad una serie di metodi Java di destinazione per acquisire sia gli input dell'utente sia il numero casuale che causa il comportamento dell'applicazione variabile nel caso in cui esso non è mantenuto deterministico tra la fase di Capture e quella di Replay. Inoltre, esso intercetta il traffico di rete collegandosi a librerie C native. Catturando, quindi, questi flussi multipli di dati ed eventi e eseguendo il timing sensitive e la riproduzione delle interazioni dell'interfaccia utente indipendente dalle coordinate, RANDR riproduce con successo il comportamento delle app Android su differenti dispositivi. L'obiettivo di RANDR è fornire funzionalità di registrazione e riproduzione su più dispositivi per le app Android il cui codice non è accessibile. RANDR non richiede né permessi di root né modifiche al sistema operativo e/o codice sorgente dell'app. D'altra parte, RANDR sfrutta il sandbox dell'applicazione Android che assicura che ogni app venga eseguita nel proprio spazio di processo all'interno della propria istanza di *runtime Android*. Dinamicamente, strumentando il framework Android dall'interno dello spazio virtuale dell'app, RANDR è in grado di acquisire e riprodurre gli input in una serie di chiamate (sia API Java che API C) al metodo target nel framework. Inoltre, RANDR si aggancia ad una serie di metodi API target per catturare gli input durante la registrazione ed iniettare gli input del registratore durante la riproduzione. L'approccio di RANDR al replay dell'interfaccia utente si basa sull'osservazione che un utente interagisce principalmente con un'applicazione tramite i widget dell'interfaccia utente, elementi essenziali dell'interfaccia utente forniti dal framework Android (per esempio, *Button, SearchBar, ImageView*, ecc). Pertanto, ha come target, il replay dei widget dell'interfaccia utente e registra gli eventi dell'interfaccia utente rispetto ai widget dell'interfaccia utente di un'app. Per identificare in maniera univoca i widget presenti in un'app, esso assegna widget identificatori stabili che sono coerenti su più applicazioni e dispositivi. RANDR deriva questi identificatori da molteplici proprietà interne di un widget, inclusa la posizione del widget nel layout dell'interfaccia utente, nel testo e in altri campi interni. Durante il replay, RANDR identifica il widget sullo schermo, aggiorna gli eventi registrati in base alla nuova posizione del widget ed inserisce gli eventi aggiornati nell'app. RANDR registra e riproduce la rete di I/O collegandosi ai metodi wrapper per varie chiamate di sistema nella libreria C di Android. Durante la registrazione, richiama il metodo della libreria originale e salva gli argomenti in un file di traccia separato per ciascun socket. Al replay, una volta chiamato il metodo su un socket, RANDR identifica il file di traccia di destinazione da leggere in base all'indirizzo IP di destinazione.

Fig2.2 Schema di RANDR

2.3 EyeAutomate

EyeAutomate è un ambiente di test freeware basato sulla libreria EyeAutomate di test visivi open source [5]. Il tool consente di automatizzare qualsiasi applicazione fornita con una GUI e che può essere emulata in un ambiente desktop, dove viene avviato l'IDE. I test vengono creati attraverso la tecnica del Capture & Replay, raccogliendo acquisizioni di immagini degli elementi grafici su cui deve essere eseguito ogni passaggio dell'interazione con la GUI. Il tool può essere visto come un'evoluzione, con più comandi integrati, del tool *Sikuli*, ed è già stato applicato in numerosi contesti industriali. EyeAutomate rientra nella categoria dei cosiddetti Visual Tool, ambienti utilizzati nella pratica industriale. Rispetto alla pratica dei test manuali, l'esecuzione dei test case con questo strumento è più veloce e la frequenza dei test è più alta. I visual tool inoltre sono capaci di trovare un quantitativo di bug superiore, se rapportato agli altri tool di C&R. Inoltre, possono aiutare a ridurre i tempi di test quando i tester devono eseguire i test di regressione. Si tratta di un tool molto flessibile in quanto può essere usato su di un qualsiasi sistema basato sulle GUI. In aggiunta, il codice dei visual tool è molto semplice e si combina con il funzionamento visivo. Un altro vantaggio dei Visual Tool è la semplicità di utilizzo da parte dei tester. Inoltre, seguire l'architettura degli script dei Visual Tool, può aiutare a scrivere i test case. Tuttavia, nonostante i numerosi vantaggi, essi presentano anche dei difetti. Tra questi, si sottolinea la difficoltà di utilizzare i Visual Tool su sistemi che non si basano sulle GUI. Essi dipendono molto dalla tecnica di riconoscimento delle immagini. Infatti, risulta difficile riconoscere immagini di dimensioni più piccole. Una volta che l'aggiornamento del software modifica il layout delle GUI, i test case devono essere modificati. Ciò ovviamente porta ad elevati costi di manutenzione. Conseguentemente, il tool non è abbastanza maturo a causa della mancanza di ricerca e pratica. D'altra parte, però, i Visual Tool hanno trovato le condizioni che aiutano le aziende ad utilizzare questo strumento come approccio di test a lungo termine. Ritornando al Visual Tool di nostro interesse, EyeAutomate, esso è un Visual script runner. Quest ultimo contiene

immagini che si possono trovare utilizzando l'algoritmo di riconoscimento delle immagini Eye. Successivamente, è possibile modificare i visual script in EyeStudio o utilizzando un Editor di testo. EyeStudio è un editor di visual script, utilizzato per creare e gestire script ed immagini. Come già accennato, EyeAutomate rientra nella cosiddetta terza generazione dei tool per il testing automatizzato delle GUI. Esso, in particolare, è un tool di Visual Testing (Visual GUI Testing – VGT). Basato sul riconoscimento delle immagini, lo strumento può effettivamente identificare visivamente la posizione sullo schermo per interagire proprio come farebbe un tester, manualmente. Ciò rende i tool utilizzati per il testing automatizzato di terza generazione insensibili sia alle coordinate fisse sia ai dettagli di implementazione dell'interfaccia utente. Esso è in grado di automatizzare qualsiasi tipo di applicazione indipendentemente dal framework e dal sistema operativo utilizzato. Uno script in EyeAutomate è visibilmente leggibile ed è composto da comandi, immagini, dati e widget.

Un visual script può essere costituito da quattro tipi di file:

- Script (estensione .txt)
- Immagini (estensione .png)
- Dati (estensione .csv)
- Widget (estensione. wid)

Di seguito, riportiamo un esempio di visual script:

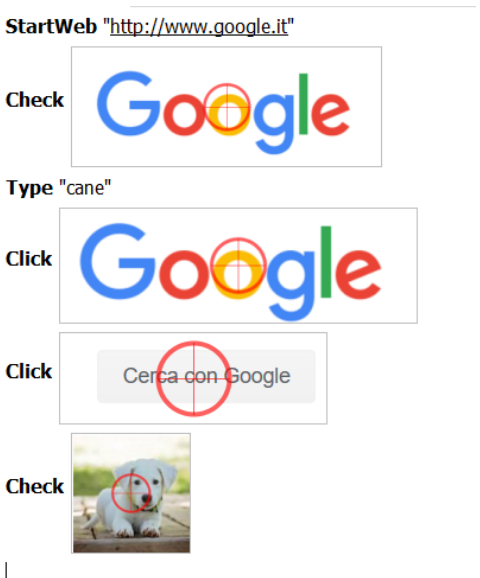


Fig2.3 Esempio di Visual Script

Esaminiamo i comandi utilizzati in questo semplice Visual Script:

- **StartWeb**, avvia il web browser predefinito con l'utilizzo dell'URL fornito (nel nostro caso, www.google.it)
- **Check,** controlla che un'immagine sia visibile. Attende fino a quando non compaia l'immagine
- **Type,** simula il testo inserito dalla testiera
- **Click,** clicca su di un'immagine

Le immagini vengono utilizzate da molti dei comandi per trovare qualcosa sullo schermo, proprio per evitare di specificare le coordinate esatte. Le immagini possono essere catturate con il Recorder oppure utilizzano il bottone *Insert/Image* in EyeStudio. Le immagini funzioneranno anche se ci sono soltanto porzioni che corrispondono effettivamente ad una parte dello schermo. La posizione di destinazione, rappresentata da un cerchio, si trova al centro dell'immagine, per default. La posizione target è più importante dei contorni.

Check 

È possibile cambiare la posizione target selezionandone un'altra all'interno dell'immagine, usando la finestra di dialogo delle proprietà. L'utilizzo di un'area di destinazione in combinazione con il comando di ricerca risulta essere molto importante per evitare di trovare qualcosa che risulti essere simile, prima di trovare l'immagine reale.

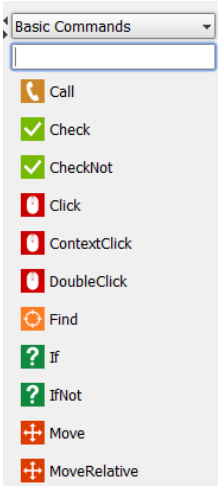


Fig2.4 Command list in EyeAutomate

Usando la Command list, è possibile selezionare un comando da inserire all'interno dello script. Inoltre, è possibile scegliere tra diverse categorie di comandi utilizzando il menù a tendina o, in alternativa, cercando un comando utilizzando il campo di ricerca. Facendo clic con il tasto destro del mouse su un comando e selezionando la guida contestuale dal menù è possibile visualizzare informazioni dettagliate, parametri disponibili ed esempi su come utilizzare il comando. Cliccando con il tasto sinistro del mouse su un comando, è possibile aggiungere quest ultimo allo script nella posizione corrente del cursore. Tuttavia, potrebbe essere necessario specificare un numero di parametri di comando prima di aggiungere il nuovo comando allo script. Un comando può essere anche digitato manualmente utilizzando l'editor. Inoltre, manualmente o utilizzando il menu dello script o i pulsanti della barra degli strumenti, è possibile aggiungere parametri, quali valori, immagini o script. Per eseguire lo script, bisogna cliccare sul bottone *Run* o selezionare l'opzione dal menù dello Script. L'output verrà visualizzato nella finestra dei log che si trova sotto allo script. È possibile anche eseguire una parte dello script: selezionando un numero di comandi e premendo il botton *Run Selection* per eseguire i comandi selezionati. Uno script può essere anche eseguito manualmente. Tutto i comandi o i passaggi vengono visualizzati come istruzioni manuali. Per eseguire manualmente uno script, basterà cliccare sul pulsante *Manual Run* nella barra degli strumenti o selezionare l'opzione *Manual Run* dal menù dello script. Il Test Report elenca tutti i comandi che sono stati eseguiti in un test. I comandi falliti avranno anche un collegamento ad uno screenshot che mostra la schermata effettiva al momento dell'errore. Verrà incluso anche uno screenshot per le immagini acquisite. Di seguito, un esempio di test Report relativo al Visual Script mostrato in precedenza:

23/06/2020 14:12:44

Command	Run Status
StartWeb http://www.google.it	Passed
Check 	Passed Screenshot
Type cane	Passed
Click 	Passed Screenshot
Click 	Passed Screenshot
Check 	Passed Screenshot

Fig2.5 Esempio di Test Report

Colonna	Descrizione
Command	Rappresenta il comando dello script.
Run status	Passed o Failed. Per i comandi *Failed* e *Capture* sarà incluso anche uno screenshot.

Fig2.6 Esempio di colonne della tabella

2.4 ODBR – On-Device Bug Reporting for Android Applications

ODBR è un tool volto a supportare tester e sviluppatori di app nel processo di segnalazione dei bug [7]. I bug che emergono nelle mobile app possono essere difficili da riprodurre e correggere a causa di numerosi fattori tra i quali la natura delle mobile app fortemente dipendente dalle GUI, differenti versioni della piattaforma e diversità dei dispositivi. È evidente quindi che gli sviluppatori abbiano

bisogno di supporto sotto forma di strumenti automatizzati che consentano un reporting più preciso dei difetti dell'applicazione al fine di facilitare la correzione dei bug, rendendola più efficiente ed efficace. ODBR è in grado di funzionare su di un dispositivo Android fisico o virtuale e di registrare precise interazioni touch dell'utente associate a specifici componenti della GUI. Successivamente, queste informazioni sono inviate ad un'applicazione Web Java che visualizza un report dettagliato dei bug, inclusi screenshot ed una serie di eventi utente che possono essere riprodotti su di un dispositivo target, consentendo a sviluppatori e tester di eseguire il debug dell'app. In commercio sono presente numerose piattaforme di analisi e segnalazione dei bug per supportare le app Android. Solitamente, queste soluzioni sono costituite da una libreria esterna che lo sviluppatore può includere durante la scrittura delle app. Questa libreria consente quindi la raccolta di informazioni, quali i log del dispositivo, informazioni sulle prestazioni, screenshot e registrazioni video. ODBR è in grado di acquisire informazioni critiche di cui uno sviluppatore ha bisogno per riprodurre e correggere con successo una segnalazione di bug.

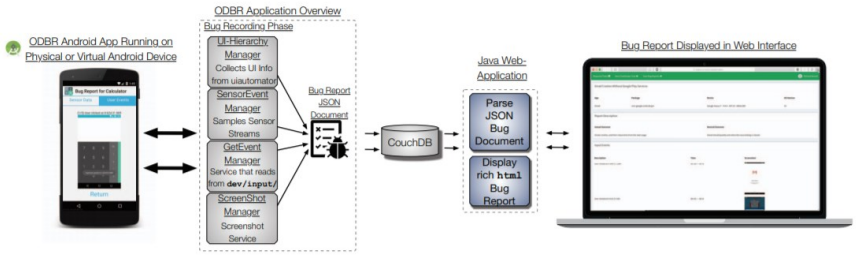


Fig2.7 Architettura di On-Device Bug Reporting

L'architettura generale di ODBR è illustrata nella Fig7. L'Entry Point per un tester o uno sviluppatore che utilizza questo strumento è l'applicazione ODBR Android, disponibile come progetto open source. Per utilizzare questo tool, il tester o lo sviluppatore deve semplicemente installare l'app ODBR sul proprio dispositivo fisico o virtuale target, selezionare l'app (dall'elenco delle applicazioni installate) per la quale si desidera segnalare un bug, premere il pulsante di registrazione e, infine, eseguire sul dispositivo le azioni che manifestano il bug. Dopo che l'app ODBR (che è in esecuzione in background) rileva che non sono stati immessi input touch dopo un certo periodo di tempo, esso chiede all'utente se ha terminato il report o se desidera continuare. Una volta che l'utente ha terminato, avrà la possibilità di inserire ulteriori informazioni sul bug, tra cui una descrizione del comportamento previsto ed effettivo e un titolo per il report. Inoltre, l'utente può visualizzare gli screenshot e le tracce dei dati del sensore e, persino, riprodurre gli eventi touch per assicurarsi che

siano stati acquisiti correttamente. Una volta che il report è creato, un BugReport viene tradotto in un documento json che viene inviato sul web ad un'istanza del server couchDB. Un'applicazione web Java legge quindi le informazioni del bug report dal file json e converte le informazioni in un report html completamente espressivo. Dall'app web Java, lo sviluppatore può visualizzare gli step di riproduzione del report, completi di screenshot e descrizioni delle azioni.

2.5 RERAN

RERAN (RECord and Replay for ANdroid) è un tool di record-and-replay nel contesto delle applicazioni Android [3]. Esso acquisisce e riproduce direttamente gli eventi di basso livello che vengono triggerati sul dispositivo. Questo approccio consente a RERAN di acquisire e riprodurre gli eventi GUI (ad esempio azioni touchscreen), nonché quelli di altri sensori di input, quali l'accelerometro, sensore di luce e la bussola. RERAN lavora in modo naturale, non invadente e senza sforzo: durante la registrazione, un componente di acquisizione legge gli eventi, senza alcun impatto evidente sull'esecuzione. Questo è in contrasto con molti approcci basati sulle parole chiave, che richiedono uno script manuale dei test case piuttosto che una raccolta diretta delle tracce. Durante la riproduzione (fase di replay), un agente di riproduzione del dispositivo invia gli eventi al dispositivo e quest'ultimo si comporta come se l'input del sensore provenisse dall'interazione dell'utente o dall'ambiente. RERAN è progettato con estrema cura per gestire il gran numero di eventi di basso livello che si verificano durante l'esecuzione e per registrarli e riprodurli con tempistiche precise. RERAN è attualmente abilitato a riprodurre i sensori i cui eventi sono resi disponibili per le applicazioni attraverso i servizi di sistema piuttosto che attraverso l'interfaccia degli eventi di basso livello (per esempio, telecamera e GPS). RERAN è utile sotto numeri aspetti. Uno di questi è rappresentato dal fatto che esso rappresenta un potente strumento di debugging in quanto può catturare e riprodurre (capture and replay) sequenze di eventi che portano a crash anomali nelle app. Ciò aiuta gli sviluppatori ad ispezionare lo stato del sistema attorno ad un punto di arresto, in sostanza, intorno ad un crash. RERAN, inoltre, ha supporto per il time warping che consente di far avanzare rapidamente l'esecuzione di determinate app al fine di raggiungere più velocemente uno stato specifico dell'app rispetto all'esecuzione originale e senza alcun input manuale aggiuntivo. Dunque, RERAN può essere utilizzato dagli sviluppatori per numerosi motivi. In generale, i tool di record and replay sono particolarmente utili per il debugging del software, per il testing e per la manutenzione. RERAN è composto da tre passaggi. I primi eventi vengono registrati con il tool *getevent* dell'SDK Android. Successivamente, l'output viene inviato al programma *Translate* di RERAN. L'output del programma *Translate* viene quindi inviato al programma Replay di RERAN. Il programma Replay rinvia gli eventi nel flusso di eventi del dispositivo.

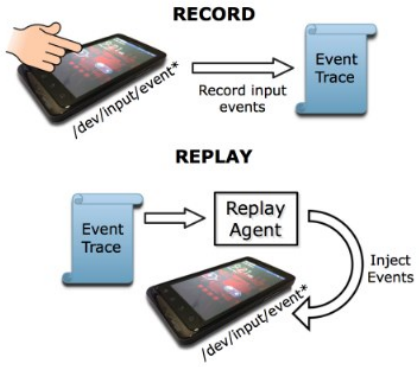


Fig2.8 Overview di RERAN

Per quanto riguarda la fase di Record, come indicato nella Fig8, viene utilizzato il tool getevent dell'SDK di Android, che legge il file */dev/input/event** , al fine di fornire un log live (una traccia) degli eventi di input del device, come mostrato nella parte superiore della Fig.8. Le prestazioni dell'app non sono influenzate durante la registrazione. Al termine della fase di logging, la fase di registrazione di RERAN traduce la traccia acquisita e crea il file di traccia dell'evento, il quale è pronto per essere riprodotto. Durante la traduzione, tutte le informazioni sugli eventi acquisiti vengono convertite in una forma concisa e vengono calcolati i ritardi. Nella fase di scrittura dei test, per motivi di implementazione, la traduzione viene eseguita offline. Sviluppi futuri prevedono di, invece, eseguire la traduzione sul device direttamente online. Per quanto riguarda, invece, la fase di Replay, Android SDK fornisce anche un tool *sendevent*, il quale consente agli sviluppatori di inviare un singolo evento di input al device. Sfortunatamente, però, il tool *sendevent* presenta un piccolo ritardo quando utilizzato in serie, il che rende impossibile riprodurre fedelmente i flussi di eventi registrati. Ci sono diverse conseguenze negative se il timing accurato non viene preservato durante la fase di Replay. Ad esempio, i ritardi nel mezzo di uno swipe (scorrimento) fanno apparire lo swipe come una serie di pressioni. Ritardando il rilascio che segna la fine della pressione, farà in modo che la pressione agirà come una pressione prolungata. La ricostruzione di uno swipe richiede, generalmente, 50 o più eventi da riprodurre. Inoltre, ogni swipe è generalmente inferiore al secondo ed il tempo tra i singoli eventi è in genere inferiore ai 60 secondi. Durante la fase di replay, un agent replay, che viene eseguito sul device ed immette gli eventi direttamente nel flusso di eventi del device, viene visualizzato come un utente esterno che genera eventi attraverso gli input del device. RERAN, eseguendo registrazione e riproduzione a livello di singoli eventi, è in grado di supportare azioni touchscreen sofisticate pur non essendo a conoscenza del significato dell'azione ad alto livello. Ciò può essere utile per trasmettere più informazioni semantiche all'utente. Tuttavia, richiede che la tassonomia di alto livello sia aggiornata ogni volta che diventa disponibile un nuovo tipo di gesto o

un nuovo tipo di sensore. Inoltre, questo approccio richiede un metodo per dedurre le azioni di alto livello dai singoli eventi di basso livello, che potrebbe non sempre essere semplice. RERAN, inoltre, è in grado di riprodurre un set di eventi di input di una specifica sessione esattamente come è stato originariamente registrato. Questo include eventi touchscreen, eventi di prossimità dell'utente, nonché cambiamenti nei sensi dell'orientamento del device da parte dell'accelerometro e della bussola. Al contrario, se un utente dovesse ripetere manualmente una sessione cercando di replicare le stesse azioni dell'esecuzione precedente, potrebbero esserci incoerenze ed anomalie rispetto all'esecuzione originale a causa di errori umani e tempistiche incoerenti. Questo, a sua volta, potrebbe condurre a risultati poco o per nulla fedeli o ad un risultato indesiderato. La fase di Replay, inoltre, può essere di grande aiuto per gli sviluppatori ed i tester che desiderano eliminare test anomali e dispendiosi, in termini di tempo, delle app, quando è necessaria la ripetizione di una frequenza.

2.6 APPIUM

Appium è un framework open source di automazione di test per applicazioni native, ibride e mobili su piattaforma iOS e Android, lato mobile, e Windows, lato desktop [21]. Le app native sono quelle scritte usando gli SDK di iOS, Android e Windows. Le mobile web app sono delle applicazioni Web a cui si accede tramite un browser mobile (Appium supporta Safari per iOS e Chrome o l'app "Browser" integrata su Android). Le app ibride hanno un wrapper intorno a una webview (si tratta di un controllo nativo che consente l'interazione con il contenuto web). Progetti come *Apache Cordova* o *Phonegap* semplificano la creazione di app utilizzando tecnologie web che vengono raggruppate in un wrapper nativo, creando quindi un'app ibrida. Soprattutto, Appium è un framework multipiattaforma: esso consente di scrive test su più piattaforme (iOS, Android, Windows) utilizzando la medesima API. Ciò consente il riutilizzo del codice tra testsuite iOS, Android e Windows. Appium è stato progettato per soddisfare le esigenze di automazione mobile secondo una filosofia delineata dai seguenti quattro principi:

1. Non è necessario ricompilare o modificare l'app in alcun modo per automatizzarla.
2. Generalmente, l'utente non è bloccato in una lingua o framework specifici per scrivere ed eseguire test.
3. Un framework di automazione mobile non dovrebbe reinventare la ruota quando si tratta di API di automazione.
4. Un framework di automazione mobile dovrebbe essere open source

Appium inoltre non ha dipendenza dal sistema operativo del dispositivo mobile dal momento che esso ha un framework o wrapper che traduce i comandi di *Selenium Webdriver* in comandi

UIAutomation (iOS) o UIAutomator (Android) a seconda del tipo di dispositivo, non del sistema operativo. Appium supporta tutte le lingue che hanno librerie client di Selenium come Java, JavaScript con node.js, PHP, Ruby, Python, C#, ecc. Appium è un "server http" scritto usando una piattaforma node.js e guida iOS e una sessione Android usando il protocollo Webdriver JSON. Quindi, prima di inizializzare il server Appium, Node.js deve essere pre-installato nel sistema. Quando Appium viene scaricato e installato, sul nostro computer viene installato un server che espose una REST API. Appium riceve la richiesta di connessione e comandi dal client ed esegue quel comando sul dispositivo mobile (Android/iOS). Esso risponde con risposte HTTP. Ancora una volta, per eseguire la richiesta, utilizza i framework di automazione dei test mobile per guidare l'interfaccia utente delle app. Una volta installato Appium, è possibile vedere la finestra di avvio del server. In quest'ultimo è presente la versione di Appium in uso, l'opzione host di default e la porta che è possibile modificare come mostrato nella seguente figura.

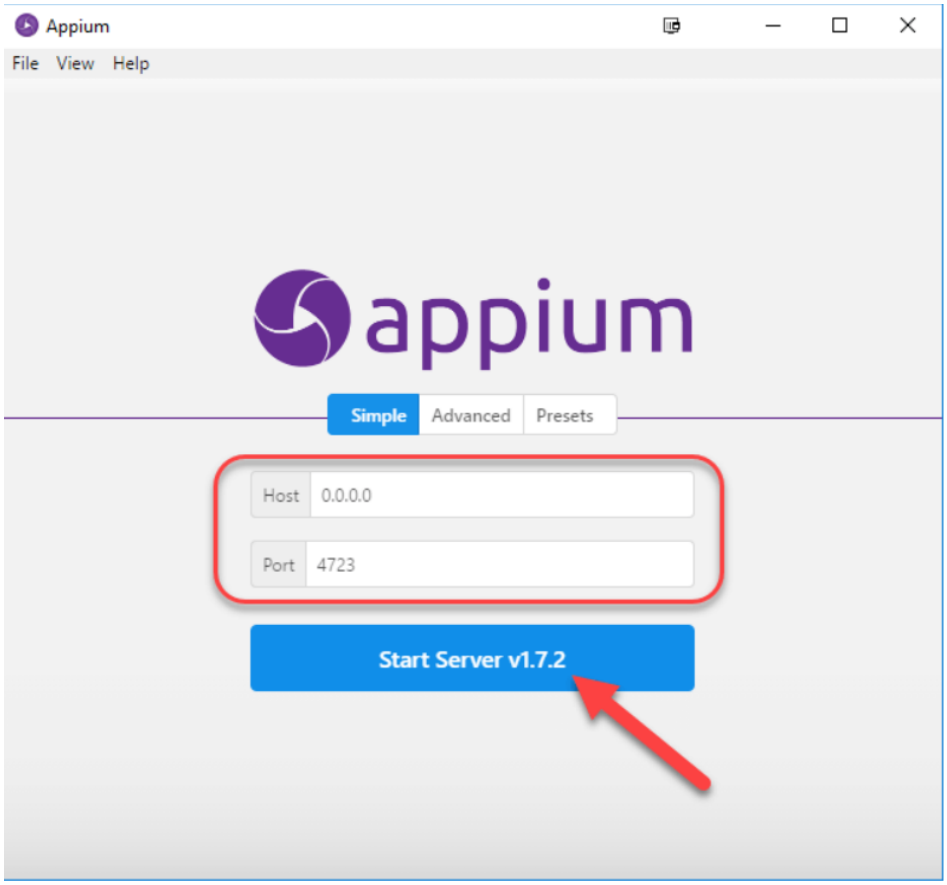


Fig2.9 Finestra di avvio del Server

Cliccando sul bottone *Start Server,* sull' host e sulla porta specificati viene avviato un nuovo server. Inoltre, vengono visualizzato i log output del server.

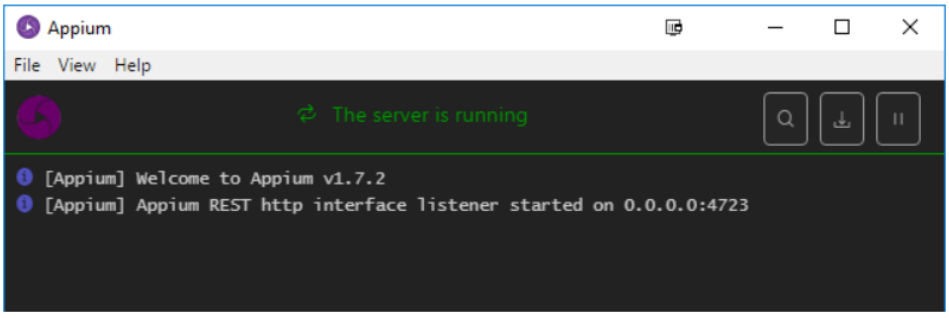


Fig2.10 Log output del server

Per cominciare una nuova sessione, bisogna cliccare su File > New Session Window.

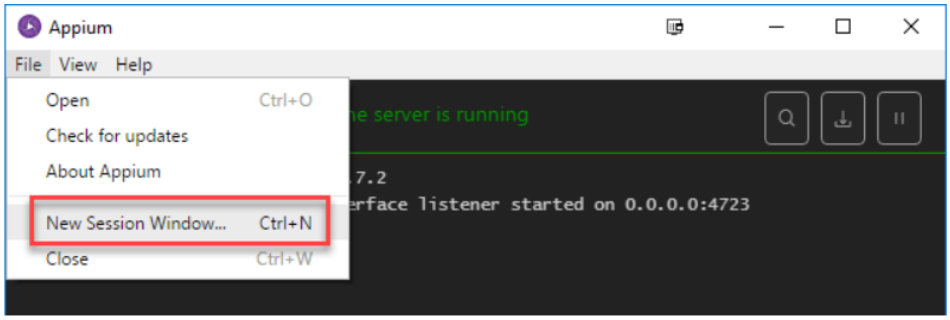


Fig2.11 New Session Window di Appium

Simile al tool Selenium IDE di registrazione (record) e riproduzione (playback), Appium presenta un Inspector per la registrazione e la riproduzione. Esso registra e riproduce il comportamento dell'applicazione nativa ispezionando il DOM e genera gli script di test in qualsiasi lingua desiderata. Tuttavia, al momento, non esiste supporto per l'Inspector di Appium per Microsoft Windows. In Windows, esso avvia il server di Appium ma non riesce ad ispezionare gli elementi. Tuttavia, UIAutomator viewer può essere utilizzato come opzione per l'ispezione degli elementi. Un'altra cosa importante per poter iniziare una sessione in Appium consiste nel definire un insieme di opzioni del server e le cosiddette Desired Capabilities. Quest'ultime rappresentano un insieme di valori e chiavi che vengono inviate all'Appium Server durante l'inizializzazione della sessione, i quali forniscono ad Appium l'informazione circa l'elemento che l'utente vuole automatizzare. L'insieme minimo di funzionalità richieste per qualsiasi driver Appium dovrebbe includere:

- *platformName:* il nome della piattaforma da automatizzare (ad esempio, Android o iOS)
- *platformVersion:* la versione della piattaforma da automatizzare
- *deviceName:* il tipo di dispositivo da automatizzare (ad esempio, Android Emulator)
- *app:* il percorso dell'app che si desidera automatizzare (ma utilizza invece la funzionalità *browserName* nel caso dell'automazione di un web browser)
- *automationName:* il nome del driver che si desidera utilizzare.

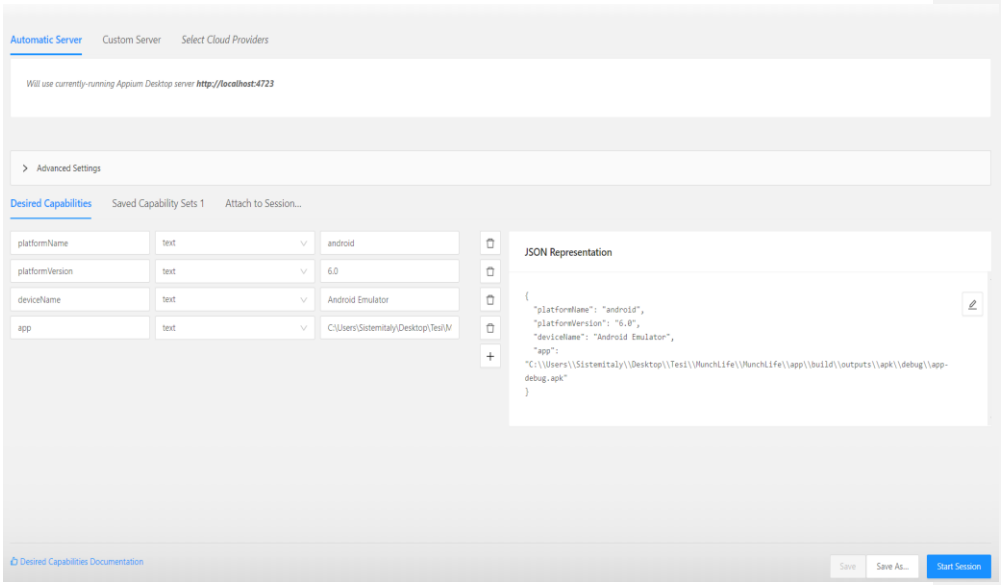


Fig2.12 Inspector di Appium con esempio di Desired Capabilities

Come si può notare dalla precedente figura, ogni client Appium sviluppa funzionalità in un modo specifico per la lingua del client, ma alla fine queste vengono inviate ad Appium come oggetti JSON. È possibile, inoltre, collegare un emulatore Android ad Appium installando l'SDK Android sul sistema. In particolare, l'utente dovrà andare in Pannello di controllo >> Sistema e sicurezza >> Sistema e cliccare, dal menù a sinistra, su "Impostazioni di Sistema Avanzate". Dalle "Proprietà di Sistema", bisogna selezionare il tab "Avanzate" e cliccare sul bottone delle "Variabili d'ambiente", come mostrato nella seguente figura.

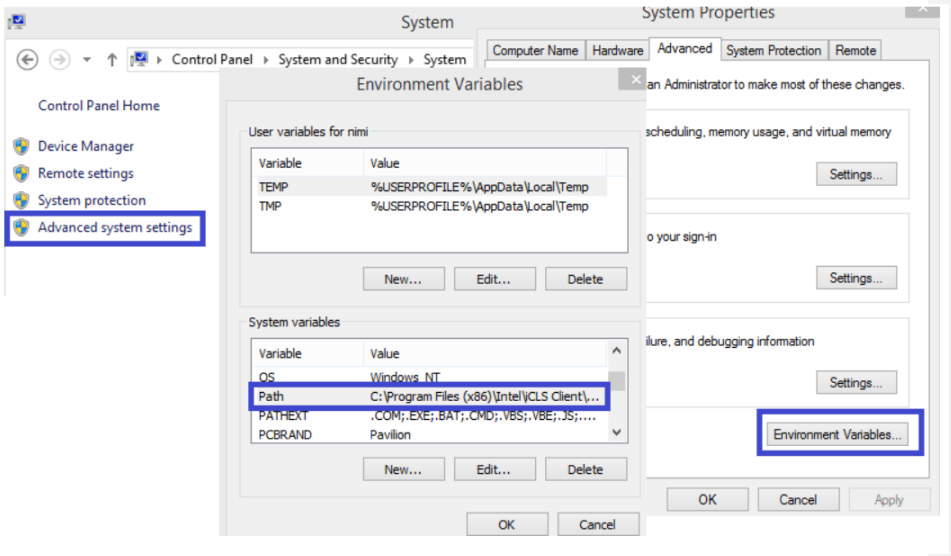


Fig2.13 Settaggio Environmental Variables

A questo punto, dalle “Variabili di Sistema”, bisogna fare doppio click sulla variabile di sistema “Path” e settare la variabile ANDROID_HOME che punta alla propria directory SDK. Nel path, bisogna aggiungere l’intero percorso della cartella SDK.

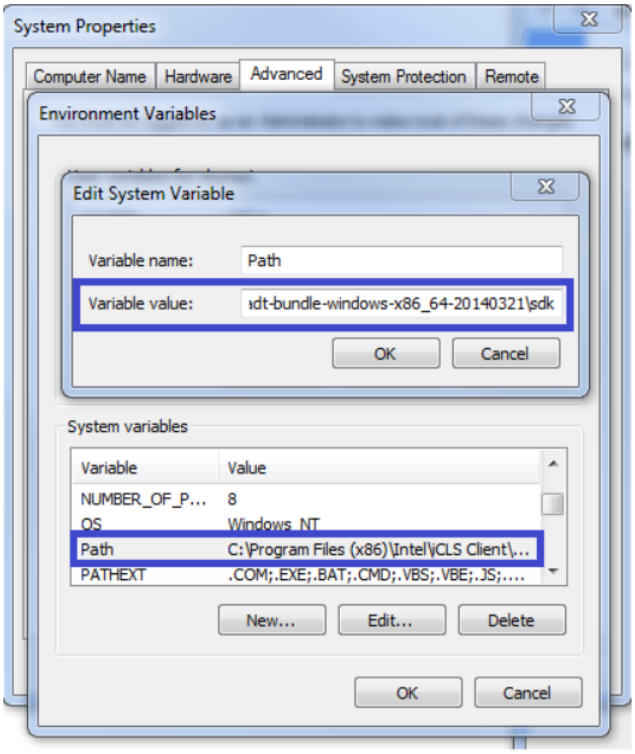


Fig2.14 Percorso della cartella SDK.

Adesso, è possibile avviare l'emulatore Android o collegare qualsiasi dispositivo Android al sistema. In quest'ultimo caso, però, bisogna assicurarsi di aver abilitato l'opzione Debugging sul dispositivo Android (per controllare l'opzione Debugging, bisogna andare in Impostazioni del dispositivo >> Opzioni Sviluppatore >> cercare "Opzione Debugging"). A questo punto, bisogna aprire il prompt dei comandi ed andare alla directory del proprio Android SDK (per esempio, nel mio caso, C:\Users\Sistemitaly\Desktop\Tesi\sdk\platform-tools). Lanciando il comando "adb devices" è possibile visualizzare i dispositivi connessi che sono "ascoltati" nella finestra del prompt dei comandi.

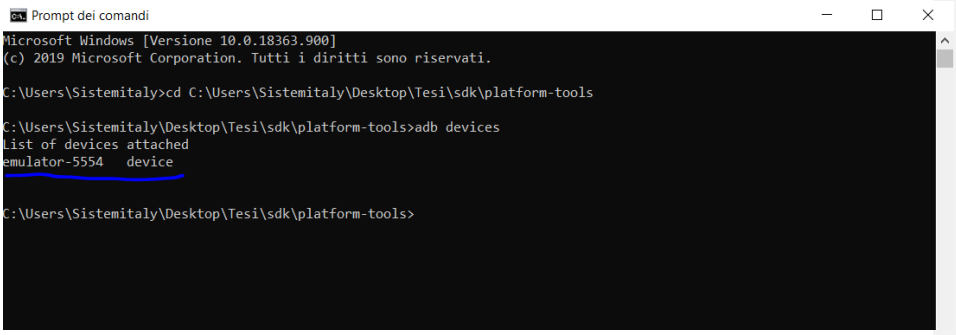


Fig2.15 Dispositivi visualizzati tramite il comando “adb devices” nel Prompt dei comandi.

Ritornando al caso della Fig7, una volta definite le Desired Capabilites, per avviare l’attività di test, basterà cliccare sul bottone “Start Session”.

2.7 Robotium

Robotium è un framework di test Android che consente di automatizzare i test case per applicazioni native ed ibride. Grazie all’aiuto di Robotium, lo sviluppatore è in grado di creare un forte caso di test automatico per le GUI per le applicazioni Android. Inoltre, lo sviluppatore potrebbe anche scrivere uno scenario di test funzionale, di sistema e di accettazione, abbracciando numerose attività di Android. Dunque, Robotium semplifica la scrittura di potenti e robusti test automatici black box dell’interfaccia utente per applicazioni Android. Robotium offre i seguenti vantaggi:

- Consente di testare applicazioni Android sia native che ibride;
- Richiede una minima conoscenza dell’applicazione in esame (Application Under Test – AUT);
- Il framework gestisce automaticamente più attività Android;
- Richiede un tempo minimo necessario per scrivere solidi test case;
- La leggibilità dei test case è notevolmente migliorata, rispetto ai test di strumentazione standard;
- I test case sono più robusti grazie all’associazione run-time ai componenti dell’interfaccia utente;
- Esecuzione rapida dei test case;
- Si integra perfettamente con Maven, Gradle o Ant per eseguire test nell’ambito di una continua integrazione.

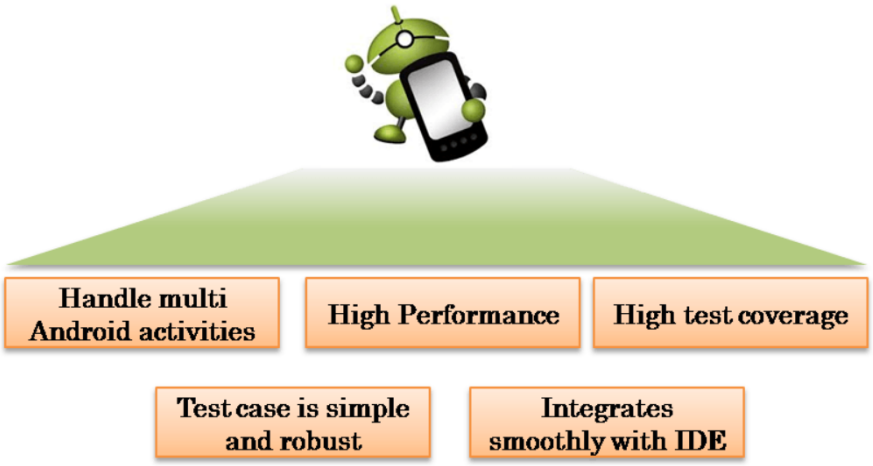


Fig2.16 Potenzialità di Robotium

Robotium utilizza una serie di classi per l'attività di testing (com.jayway.android.robotium.solo). Questa classe supporta test case che si estendono su più attività. Solo è integrato con ActivityInstrumentationTestCase2. Il tester può scrivere test case senza conoscere la progettazione dell'applicazione (testing back box) utilizzando le classi di test case di Robotium.

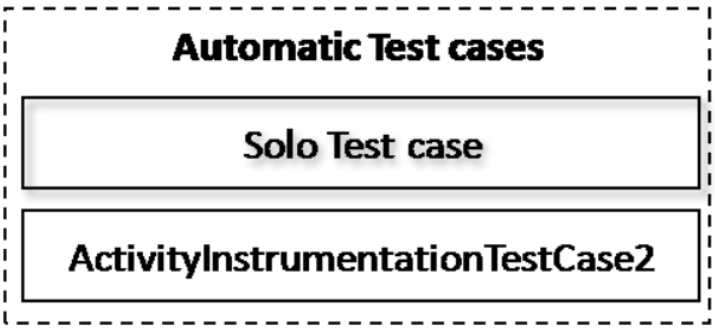


Fig2.17 Integrazione Robotium e ActivityInstrumentationTestCase2

Prima di cominciare con la fase di Record, bisogna essere certi di aver installato il JDK di Java e l'SDK di Android. In Android Studio, è possibile installare Robotium Recorder come plugin. Una volta scaricato, nella barra degli strumenti di Android Studio, troveremo Robotium Recorder nel tab Tools. Cliccando sul tab Tools, è possibile visualizzare la seguente schermata:

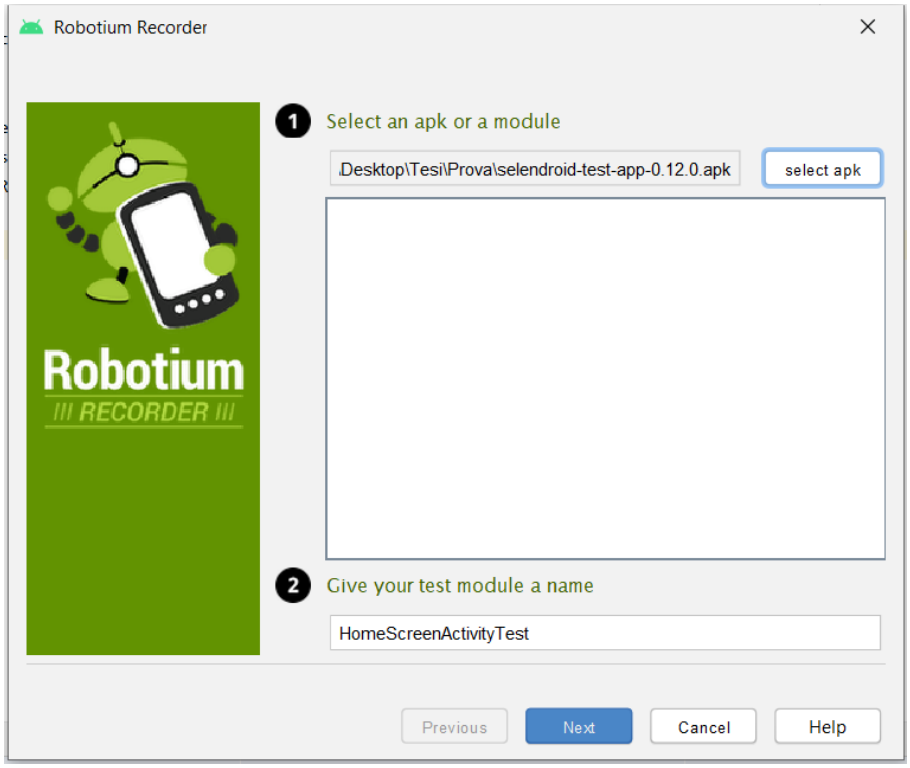


Fig2.18 Robotium Recorder in Android Studio

Nella schermata di avvio di Robotium, viene chiesto dapprima di selezionare l'apk o un modulo (punto 1) e, successivamente, di dare un nome al test che si intende eseguire (punto 2). Dopo aver cliccato su *Next*, come mostrato nella figura precedente, viene visualizzata la finestra "Start the recording":

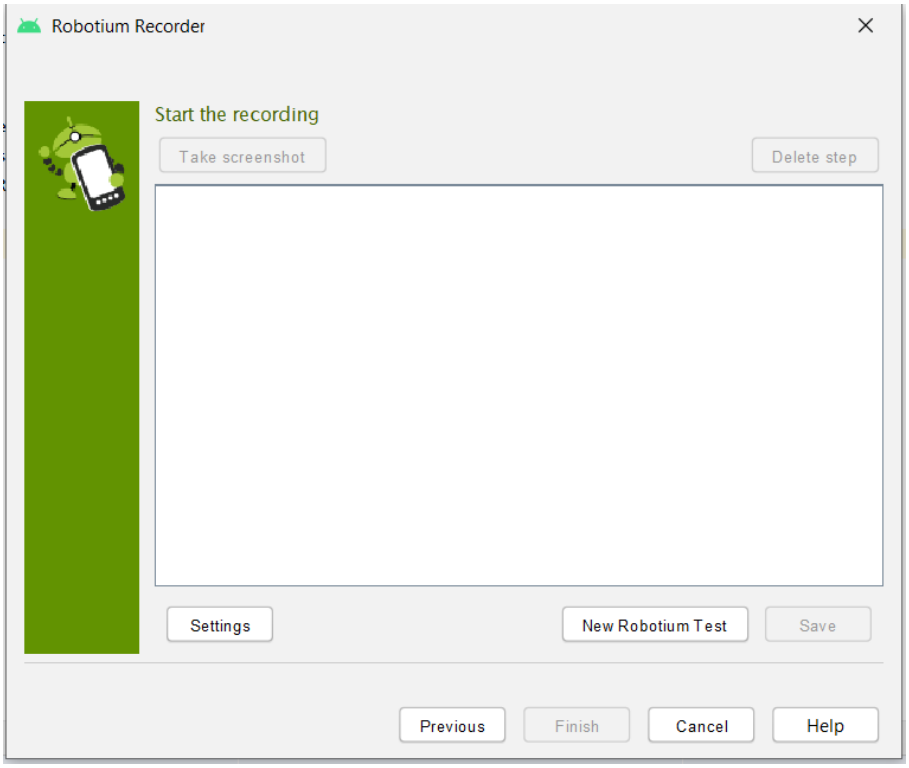


Fig2.19 Finestra di "Start the recording"

In questa finestra, vi sono numerosi comandi. C'è il bottone *New Robotium Test* (abilitato), il bottone *Save* per salvare una sessione (disabilitato), un bottone *Take Screenshot*, per fare uno screenshot (disabilitato), ed un bottone *Delete Step* per eliminare uno step registrato, anch'esso disabilitato. Tutti i bottoni disabilitati diventano attivi durante la sessione di registrazione o al termine di essa. Nell'angolo in basso a sinistra della finestra di "Start the recording" come mostrato nella Fig19, è presente un bottone *Settings* che fornisce le seguenti tre opzioni:

1. *Use sleeps* – Questa opzione può essere selezionata per utilizzare la modalità sleep in riproduzione. Il test case verrà eseguito contemporaneamente. L'app è stata registrata. Si tratta di una funzionalità interessante soprattutto per le applicazioni che consumano molta larghezza di banda.
2. *Keep app data* – I dati dell'app verranno conservati quando si avvia un nuovo record di test.
3. *Identify class over string* – Con questa opzione, l'identificatore di visualizzazione predefinito, l'id della risorsa vengono utilizzati al posto dell'identificatore di stringa.

Per iniziare una nuova registrazione, bisogna cliccare sul bottone *New Robotium Test*, come indicato nella Fig19. Robotium Recorder installerà ed eseguirà l'APK sul dispositivo connesso o sull'emulatore. Ogni touch, scorrimento, rotazione e digitazione verrà registrato e potrà essere visualizzato nel wizard di Robotium Recorder. Mentre il recorder è in esecuzione, l'utente ha la possibilità di inserire uno screenshot o di eliminare uno step appena fatto. Per abilitare entrambi i comandi, è necessario fare clic sugli step appena registrati. Se si desidera interrompere la registrazione, è sufficiente cliccare sul pulsante di *Stop Recording*. Dopo aver interrotto la registrazione, verrà abilitato il pulsante *Save*. In questa fase, si ha ancora la possibilità di eliminare uno step o di aggiungere uno screenshot nelle registrazioni. Quando si preme il bottone *Save*, viene visualizzata una finestra di dialogo per l'immissione di un nome. In conclusione, quindi, possiamo dire che Robotium è il tool di automazione di test Android più comunemente utilizzato. I test case Robotium possono essere eseguiti sull'emulatore Android o su un dispositivo reale. Non è necessario eseguire, inoltre, alcun codice di configurazione specifico per eseguire test case Robotium sul dispositivo reale. Infine, Robotium è molto utile per scrivere script di test Android di automazione facili e semplici. Come già detto, si tratta di un tool open source che, però, fornisce 10 test gratuiti. I successivi necessitano dell'acquisto di una licenza. D'altra parte, nonostante i vantaggi dell'utilizzo di Robotium, quest'ultimo presenta, come ogni altro strumento, degli svantaggi. In primis, non è in grado di gestire componenti Flash o Web; gestisce una sola applicazione alla volta. Non è possibile simulare il clic sulla tastiera utilizzando Robotium. È un tool che può risultare lento specialmente se eseguito su dispositivi piuttosto vecchi (meno recenti). Inoltre, Robotium non è in grado di interagire con le notifiche della barra di stato, nonché trascinare verso il basso l'area di notifica e fare clic su una specifica notifica.

2.8 Selendroid

Selendroid è un framework di automazione di test che si allontana dall'interfaccia utente delle applicazioni Android native, ibride ed applicazioni web.

Fig2.20 Applicazioni che Selendroid consente di testare

È possibile scrivere i test utilizzando 2 API client di Selenium dal momento che Selendroid riutilizza ancora l'infrastruttura di Selenium esistente per il Web. Ricordiamo infatti che Selendroid non è altro che il tool Selenium per applicazioni Android. Selendroid è un potente strumento di test. Può essere utilizzato sia su emulatori che su dispositivi reali. Inoltre, può essere integrato come come nodo nel Grid di Selenium per il testing parallelo. Selendroid presenta numerosi vantaggi:

- È possibile testare l'applicazione che si intende esaminare (AUT) con l'ausilio di Selendroid senza alcuna modifica dell'app. Si ha solo bisogno del file binario (l'APK) installato sul proprio computer. Per installare il file binario sul dispositivo, l'app di test e l'app mobile devono essere entrambe firmate con la stessa chiave.
- L'app di test Selendroid può interagire con più dispositivi o simulatori contemporaneamente. Ciò rappresenta un enorme vantaggio. Quindi, è possibile testare l'app di interesse con vari dispositivi Android per verificarne la compatibilità.
- Selendroid è in grado di simulare le azioni umane dell'utente su un'applicazione, come il touch, lo scorrimento, il trascinamento sui dispositivi.
- È possibile modificare i dispositivi hardware durante i test senza riavviare o interrompere gli stessi. Selendroid riconosce automaticamente i nuovi dispositivi.
- Selendroid, inoltre, ha anche un tool Inspector che aiuta ad identificare l'elemento dell'interfaccia utente dell'applicazione in esame (AUT). Ad esempio, l'ID button, il campo di testo e così via.

Selendroid si basa sull' Android instrumentation framework. I test di Selendroid sono scritti sulla base dell'API client del web driver di Selenium, quindi supporta la piena integrazione con gli attuali framework di Selenium.

La seguente figura descrive a pieno l'architettura di Selendroid:

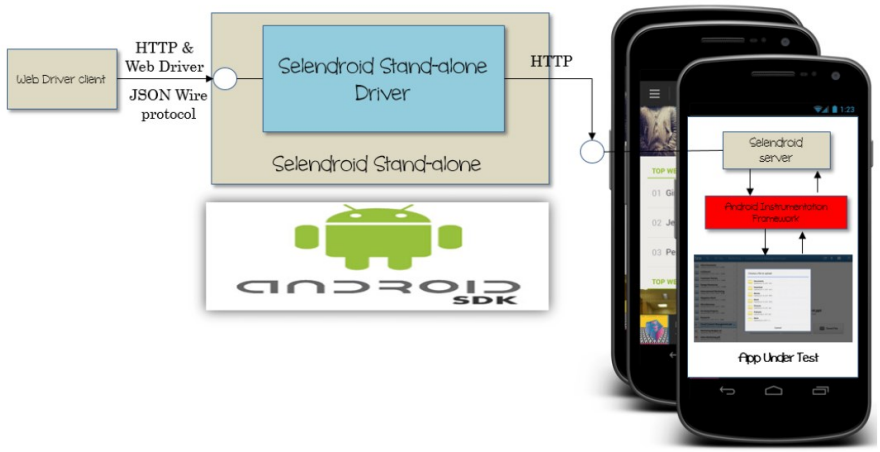


Fig2.21 Architettura di Selendroid

Selendroid, inoltre, contiene quattro componenti principali:

1. *Web Driver Client* – La libreria client Java basata su Selenium. Questa libreria dovrebbe essere installata sul computer (la quale viene poi utilizzata per sviluppare i test case).
2. *Selendroid-Server* – Si tratta del server in esecuzione nell'applicazione in esame (AUT) sul dispositivo Android o sull'emulatore. Questo è il componente principale dell'architettura di Selendroid.
3. *Android Driver-App* – Un driver Android integrato, un'applicazione di visualizzazione web che consente di testare il mobile web.
4. *Selendroid-Standalone* – Questo componente viene utilizzato per installare il server Selendroid e l'applicazione in esame (AUT).

Come già citato, Selendroid può essere utilizzato per testare applicazioni già costruite. Queste applicazioni Android (nonché gli APK) devono essere presenti sulla macchina, nella quale verrà avviato il server *selendroid-standalone*. Il motivo di ciò è che verrà creato un *Selendroid-Server* customizzato per l'app in esame (AUT). Entrambe le app (Selendroid-Server e AUT) devono essere firmate con il medesimo certificato per installare gli APK sul dispositivo. Per eseguire Selendroid, è necessario installare l'SDK di Android e il JDK di Java; avere a disposizione sul computer almeno un dispositivo virtuale (AVD) o un dispositivo reale Android; il tool Eclipse; Selendroid-Standalone,

Selendroid Client e Selendroid Client. Selendroid è un framework open source che ci consente di automatizzare test case per applicazioni mobili. Esso presenta numerosi vantaggi tra i quali la possibilità di automatizzare numerose interazioni dell'utente su dispositivi mobili. Tuttavia, ad oggi, il lavoro di sviluppo ed il supporto online sono quasi completamente interrotti. È possibile trovarsi di fronte ad alcuni problemi durante l'esecuzione di alcune azioni.

2.9 Espresso Test Recorder

Android Record Espresso Test è un'estensione di Android Studio, tool più grande. Esso consente di creare test UI senza scrivere alcun codice di test [11]. Registrando uno scenario di test, è possibile registrare le interazioni con un dispositivo, reale o virtuale, ed aggiungere delle asserzioni per verificare gli elementi dell'interfaccia utente (UI), in particolare gli snapshots dell'app. Espresso Test Recorder prende quindi la registrazione salvata e genera automaticamente un test UI corrispondente che, successivamente, l'utente può eseguire per testare la sua app. Dunque, per utilizzare Espresso Test Recorder sarà sufficiente scegliere l'opzione Record Espresso test ed indicare una macchina reale o virtuale; eseguire una sequenza di eventi sull'interfaccia della macchina Android (il pannello mostrerà un riassunto di tale sequenza) ed aggiungere una o più asserzioni. In quest'ultimo caso, il tool aiuterà il tester a riconoscere e definire elementi dell'interfaccia rispetto ai quali basare le asserzioni. I test Android Espresso consistono in due componenti principali: interazioni dell'interfaccia utente e ed asserzioni sugli elementi visualizzati. Le interazioni dell'interfaccia utente includono le azioni "tocca e digita" che una persona può utilizzare per interagire con l'app. Le asserzioni, invece, verificano l'esistenza o il contenuto degli elementi visivi a schermo. Ad esempio, un test Android Espresso per l'app di Notes potrebbe includere interazioni dell'interfaccia utente per fare click su di un pulsante e scrivere una nuova nota, ma utilizzerebbe asserzioni per verificare l'esistenza del pulsante ed il contenuto della nota. A questo punto, vediamo come creare entrambi questi componenti di test, le interazioni e le asserzioni, utilizzando Espresso Test Recorder e come salvare la registrazione finita per generare il test.

2.9.1 Interazioni interfaccia utente

Per cominciare a registrare un test con Espresso Test Recorder, bisogna seguire i seguenti passi:

1. Cliccare su **Run** > **Record Espresso Test**

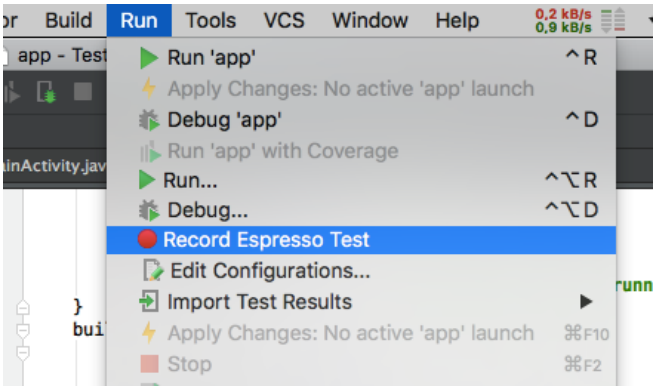


Fig 2.22 Record Espresso Test

2. Nella finestra **Select Deployment Target**, bisogna scegliere il device sul quale si vuole registrare il test. Se necessario e, nel caso in cui non è stato fatto prima, bisogna creare un nuovo AVD (Android Virtual Device).

3. Espresso Test Recorder avvia una build del progetto e l'app deve essere installata e lanciata prima che Espresso Test Recorder consenta di interagire con esso. La finestra "Record your test" appare dopo l'avvio dell'app e, dal momento che l'utente non ha ancora interagito con il dispositivo, il pannello principale riporta "No event reported yet". A questo punto, bisogna cominciare ad interagire con il dispositivo per avviare la registrazione degli eventi, come azioni del tipo "tap" e "type", ovvero, "tocca" e "digita".

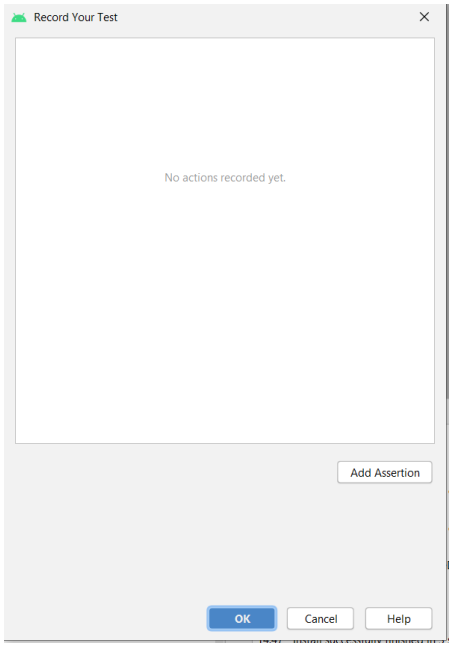


Fig 2.23 La finestra "**Record Your Test**".

Prima, però, di poter iniziare a registrare le interazioni, sul dispositivo potrebbe essere visualizzata una finestra di dialogo nella quale viene riportato il seguente messaggio: "Waiting for debbuger" o "Attaching debbuger". Espresso Test Recorder utilizza il debugger per registrare gli eventi dell'interfaccia utente. Allorchè il debbuger si collega, la finestra di dialogo si chiuderà automaticamente. Non bisogna premere sul pulsante "Force Close" pena la chiusura del processo in corso.

Dopo aver cominciato ad interagire con il dispositivo mediante azioni del tipo "tap" e "type" , le interazioni registrate verranno visualizzate nel pannello principale nella finestra "Record your Test", come mostrato nella seguente figura :

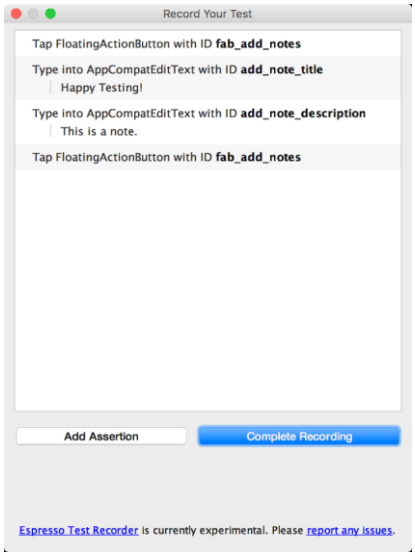


Fig 2.24 La finestra "**Record Your Test**" con le interazioni dell'interfaccia utente registrate.

Quando si esegue il test, il test Espresso proverà ad eseguire le azioni elencate nel medesimo ordine.

2.9.1.1 Aggiungere Asserzioni per verificare gli elementi dell'interfaccia utente.

Le asserzioni verificano l'esistenza o il contenuto di un elemento visivo tramite tre tipologie principali:

- **text is** : controlla il contenuto del testo dell'elemento View selezionato
- **exists** : controlla che l'elemento View sia presenta nella gerarchia View corrente visibile sullo schermo
- **does not exist** : l'opposto del precedente. Verifica che l'elemento View non sia presente nella gerarchia View corrente.

Per aggiungere un'asserzione al test, bisogna:

1. Cliccare **Add Assertion.** Viene visualizzata una finestra di dialogo **Screen Capture** mentre Espresso Test ottiene la gerarchia dell'interfaccia utente ed altre informazioni sullo stato corrente dell'app. La finestra di dialogo si chiude automaticamente dopo che Espresso ha catturato lo screenshot.

2. Un layout della schermata corrente appare in un pannello a destra della finestra Record your Test. Per selezionare un elemento View sul quale creare un'asserzione, bisogna cliccare sull'elemento nello screenshot o, in alternativa, utilizzare il primo menù a discesa nella casella **Edit Assertion** nella parte inferiore della finestra. L'oggetto View selezionato è evidenziato in una casella rossa.
3. A questo punto, bisogna selezionare l'asserzione che si desidera utilizzare dal secondo menù a discesa nella casella **Edit Assertion**. Espresso Test Recorder popola il menù con affermazioni valide per l'elemento View selezionato.
    - Se si sceglie l'asserzione "**text is**" , Espresso Test Recorder inserisce automaticamente il testo attualmente all'interno dell'elemento View selezionato. Si può modificare il testo in modo che corrisponda all'asserzione desiderata utilizzando il campo di testo nella casella Edit Assertion.
4. Infine, bisogna cliccare su **Save and Add Another** nel caso in cui si voglia salvare ed aggiungere un'altra asserzione o, in alternativa, cliccare su **Save Assertion** per chiudere il pannello delle asserzioni.

Il seguente screenshot mostra un'asserzione text is creata per verificare che il titolo della nota sia "Hello Word".

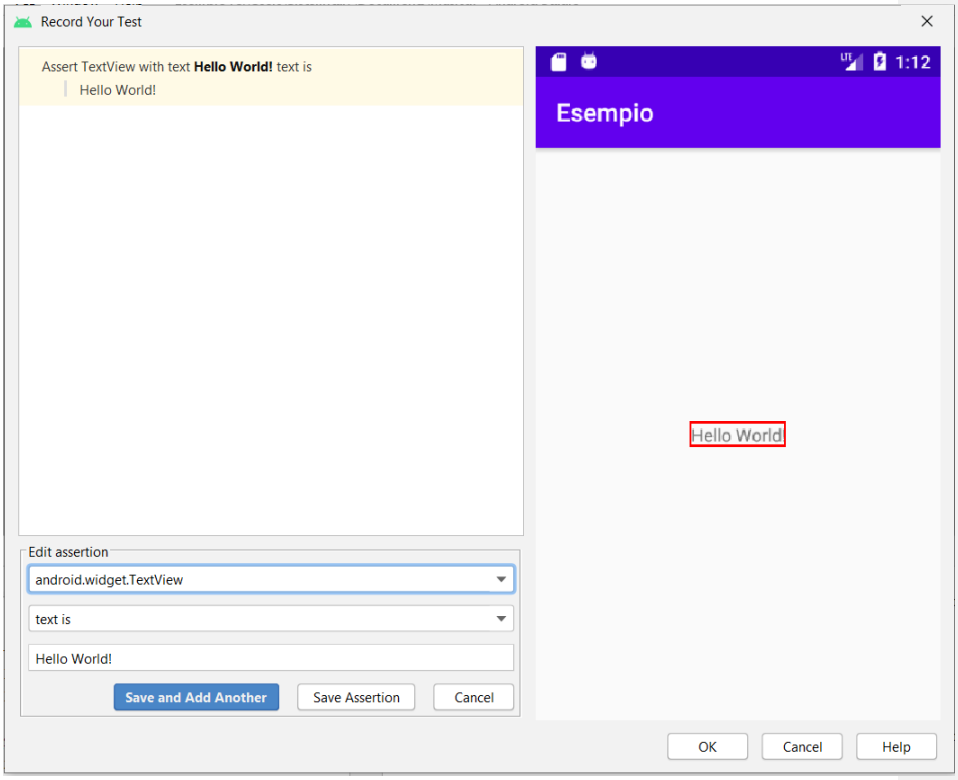


Fig 2.25 La casella **Edit assertion** dopo aver selezionato un elemento View (in rosso).

Durante la creazione di un'asserzione, l'utente può continuare ad interagire con l'app, anche con i pannelli delle asserzioni ancora aperti nella finestra Record Your Test. Espresso Test Recorder continuerà a registrare le azioni dell'utente, ma l'asserzione che egli sta modificando verrà visualizzata prima di queste interazioni una volta salvata. Lo screenshot per l'asserzione mantiene anche il layout del dispositivo o dell'emulatore al momento in cui si preme il pulsante Add Assertion.

2.9.1.2 Salvare una registrazione

Una volta terminata l'interazione con l'app e l'aggiunta di asserzioni, bisogna attenersi ai seguenti step per salvare la registrazione e generare il corrispondente Test Espresso:

Una volta cliccato su ok, viene visualizzata una finestra denominata Specify a test class for your test.

1. Espresso Test Recorder assegnerà al test un nome univoco all'interno del suo pacchetto in base al nome dell'attività avviata. Il campo di testo **Test class name** può essere utilizzato per nel caso in cui si voglia modificare il nome suggerito. Mentre, il campo **Test class language** consente di scegliere tra due linguaggio di programmazioni utilizzati in Android Studio, ovvero Java e Kotlin. Al termine delle scelte effettuate, bisogna premere su ok.
    - Nel caso in cui non siano state aggiunte all'app le dipendenze Espresso, viene visualizzata a video una finestra di dialogo Missing Espresso dependencies quando si tenta di salvare il test. In tal caso, bisognerà premere su ok per aggiungere
      automaticamente le dipendenze al file build.gradle.

2. Il file si apre automaticamente dopo che Espresso Test Recorder l'ha generato e Android Studio mostra la classe di test selezionata nella finestra Project dell'IDE.
    - Il punto nel quale viene salvato il test dipende dalla posizione della radice del test di strumentazione e dal nome del pacchetto dell'attività avviata.

Una premessa sulla quale andrebbe spesa qualche parola è la seguente: prima di utilizzare Espresso Test Recorder, bisogna assicurarsi di disattivare le animazioni sul dispositivo di test. Lasciare le animazioni di sistema attivate nel dispositivo di test potrebbe causare risultati imprevisti o il fallimento del test. Dunque, bisogna disattivare le animazioni dai Settings mediante la sezione Developer options e disattivando le seguenti opzioni:

- **Window animation scale**
- **Transition animation scale**
- **Animator duration scale**

2.9.2 Creare e gestire dispositivi virtuale – AVD (Android Virtual Device)

Un dispositivo virtuale Android, indicato anche con l'acronimo AVD, è una configurazione che definisce le caratteristiche di un dispositivo Android, tablet, sistema operativo Wear, Android TV o dispositivo OS automobilistico che si desidera simulatore nell'emulatore Android. AVD Manager è

un'interfaccia che può essere avviata da Android Studio, la quale aiuta a creare e gestire i dispositivi AVD.

Per aprire AVD Manager, è possibile eseguire una delle seguenti opzioni:

- Selezionare **Tools > AVD Manager**.
- Fare click sull'icona **AVD Manager** nella toolbar, barra degli strumenti.

Fig 2.26 Android Virtual Device Manager

Un AVD contiene un profilo hardware, un'immagine di sistema, un'area di archiviazione, skin e altre proprietà. Generalmente, bisognerebbe creare un AVD per ogni immagine di sistema che l'app di test potrebbe potenzialmente supportare in base all'impostazione <uses-sdk> del bundle manifest.

2.9.2.1 Profilo Hardware

Il profilo hardware definisce le caratteristiche di un dispositivo alle sue impostazioni di fabbrica. AVD Manager viene precaricato con determinati profili hardware, come i dispositivi Pixel, ed è possibile definire o personalizzare i profili hardware in base alle esigenze.

2.9.2.2 Immagini di sistema

Un'immagine di sistema etichettata con API Google include l'accesso ai servizi di Google Play. Un'immagine di sistema etichettata con il logo di Google Play nella colonna Play Store include l'app Google Play Store e l'accesso e l'accesso ai servizi di Google Play, includendo una scheda Google Play nella finestra di dialogo Extended Controls che fornisce un comodo pulsante per

l'aggiornamento dei servizi di Google Play sul dispositivo. Per garantire la sicurezza delle app e un'esperienza coerente con i dispositivi fisici, le immagini di sistema con Google Play Store incluso sono firmare con una release key, il che significa che non è possibile ottenere privilegi elevati (root) con queste immagini. Nel caso in cui si avesse bisogno di privilegi elevati (root) nella risoluzione di problemi con le app, è possibile utilizzare le immagini di sistema Android Open Source Project (AOSP) che includono app o servizi Google.

2.9.2.3 Memoria

AVD ha un'area di archiviazione dedicata sulla macchina di sviluppo. Memorizza i dati dell'utente del dispositivo, come le app e le impostazioni installate, nonché una scheda SD emulata. Se necessario, è possibile utilizzare AVD Manager per cancellare i dati utente, in modo che il dispositivo abbia gli stessi dati come se fossero nuovi.

2.9.2.4 Skin

Una skin dell'emulatore specifica l'aspetto del dispositivo. AVD Manager fornisce alcune skin predefinite. È possibile anche utilizzare skin fornite da terze parti, oppure la propria.

2.9.2.5 Creare un AVD

Per creare un AVD:

1 - Bisogna aprire AVD Manager cliccando su **Tools > AVD Manager**

Fig 2.27 AVD manager – Your Virtual Devices

2 – Cliccare su **Create Virtual Device** nella parte inferiore della finestra di dialogo AVD Manager. A questo punto, la pagina **Select Hardware** compare

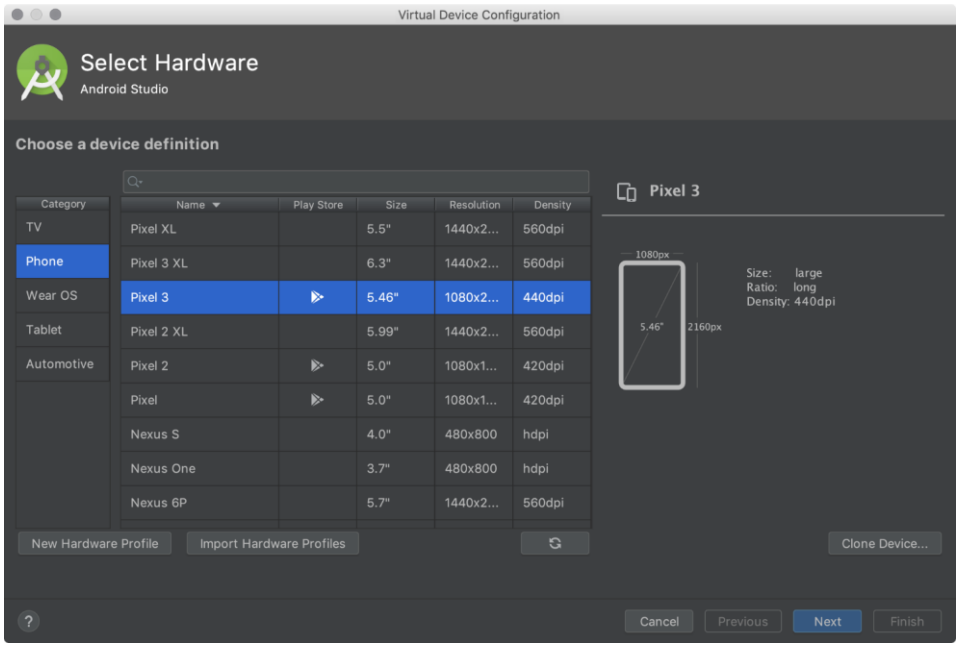


Fig 2.28 Select Hardware dell'AVD Configuration.

Si noti come solo alcuni profili hardware sono indicati per essere inclusi nel Play Store. Ciò indica che questi dispositivi sono completamente conformi a CTS e possono utilizzare immagini di sistema che includono l'app Play Store.

3 – Selezionare un profilo Hardware e premere su **Next.**

Nel caso in cui non si veda il profilo hardware desiderato, è possibile creare o importare un profilo hardware. A questo punto, la System Image compare.

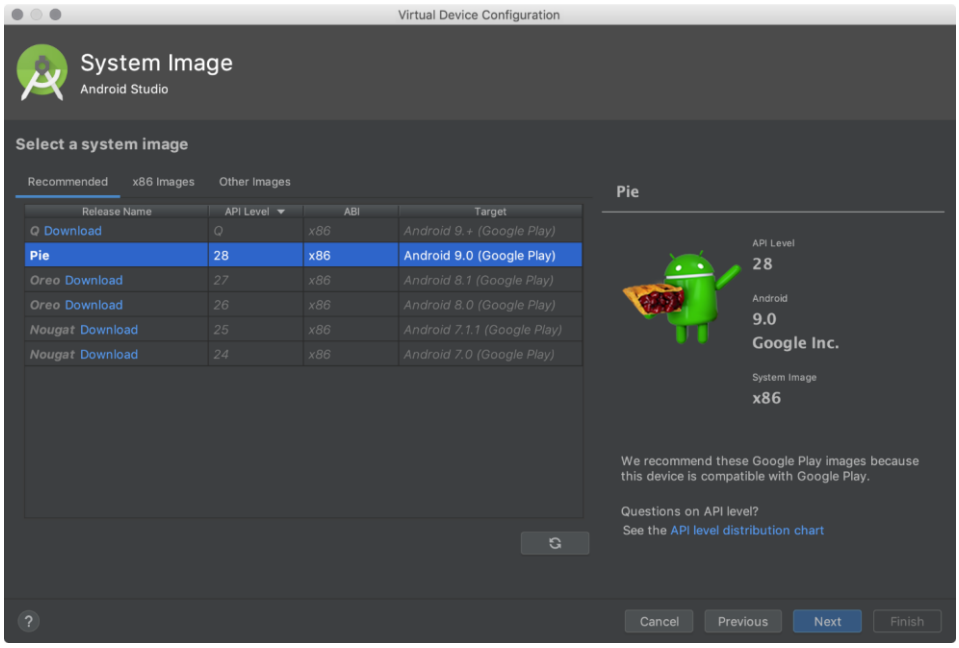


Fig 2.29 La finestra "**System Image**"dell'AVD Manager.

4 – Selezione l'immagine di sistema per uno specifico API, quindi premere su **Next**.

Il tab **Recommended** elenca le immagini di sistema consigliate. Le altre schede includono un elenco più completo. Il riquadro destro descrive l'immagine di sistema selezionata. Le immagini x86 sono le più veloci nell'emulatore. Se si clicca su Download accanto all'immagine di sistema, bisogna fare click su di esso per scaricare l'immagine di sistema. Tuttavia, bisogna ovviamente essere connessi ad internet per poterlo scaricare. L'API level del dispositivo di destinazione è importante dal momento che l'app non sarà in grado di funzionare su un'immagine di sistema con un livello API inferiore a quello richiesto dall'app, come specificato nell'attributo minSdkVersion del bundle manifest dell'app. Se l'app dichiara un elemento <uses-library> nel bundle manifest, l'app richiede un immagine di sistema in cui è presente quella libreria esterna. Se si vuole eseguire l'app sull'emulatore, bisogna creare un AVD che includa la libreria richiesta. Per fare ciò, potrebbe essere necessario utilizzare un componente aggiuntivo per la piattaforma AVD; ad esempio, il componente aggiuntivo API di Google contiene la libreria Google Maps.

A questo punto, la pagina **Verify Configuration** compare.

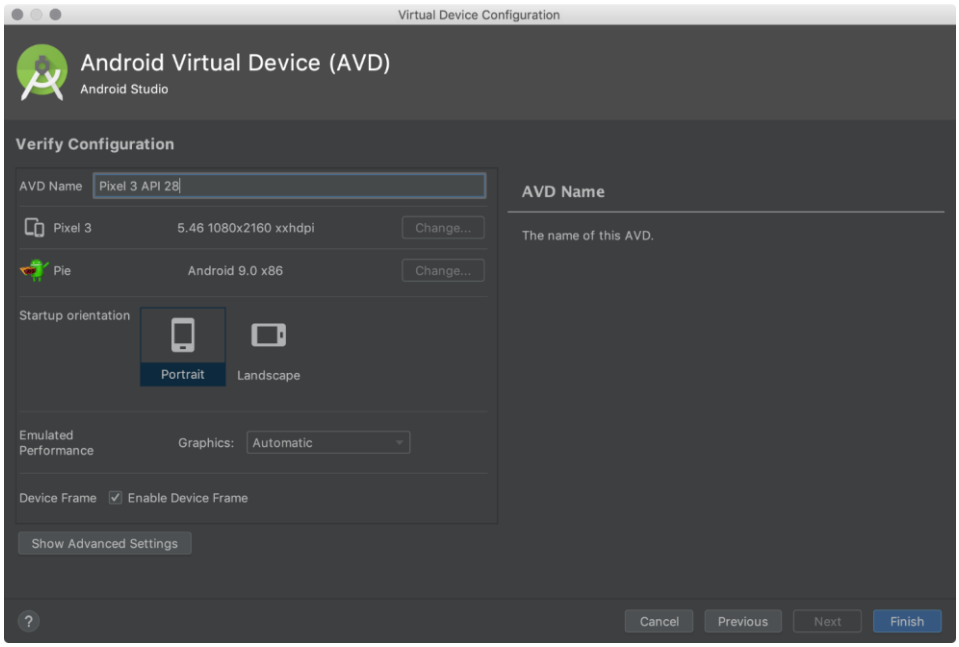


Fig 2.30 Android Virtual Device (AVD).

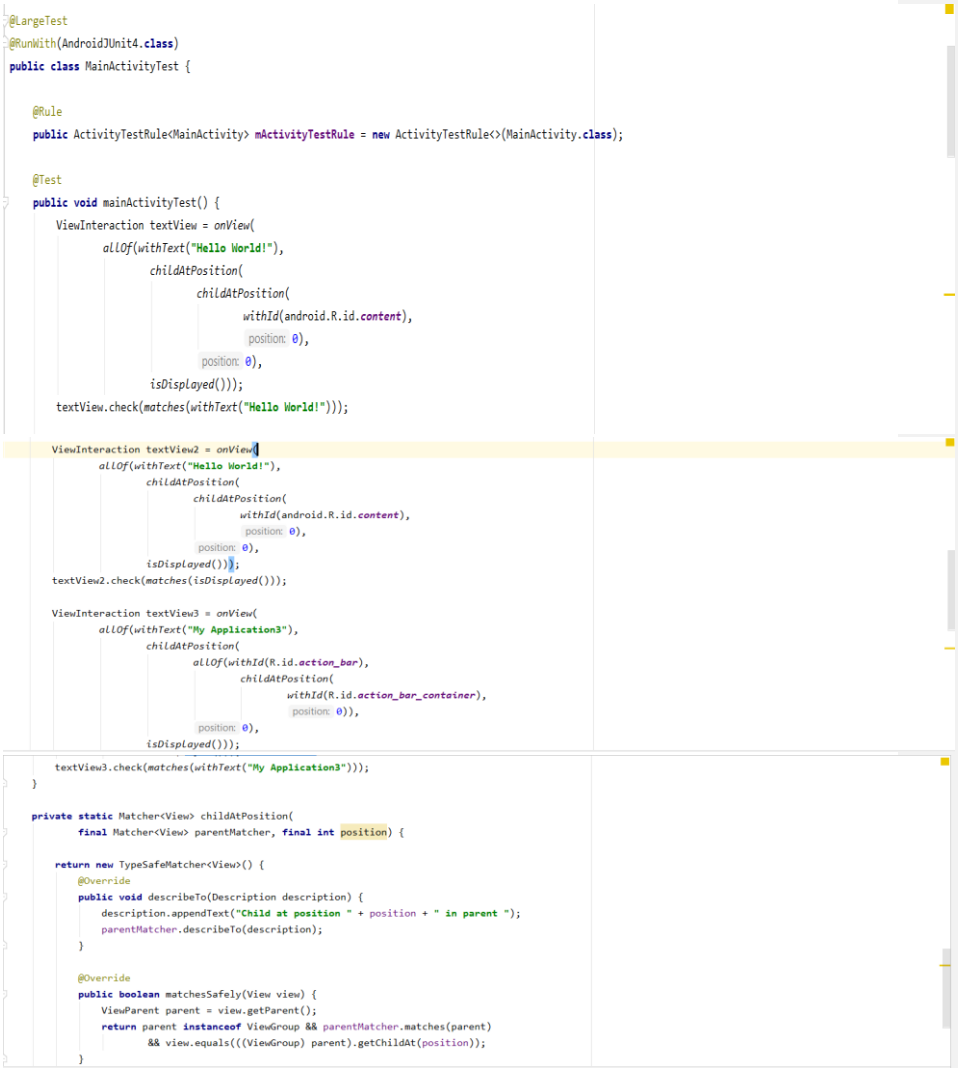
5 – Modifcare le proprietà AVD in base alle esigenze, quindi cliccare su **Finish.**

Per mostrare maggiori informazioni, come la skin, bisogna cliccare su **Show Advanced Settings.** Il nuovo AVD viene visualizzato nella pagina **Your Virtual Devices** o nella finestra di dialogo **Select Deployment Target.**

È possibile, inoltre, duplicare il proprio AVD:

- Dalla pagina **Your Virtual Devices** dell'AVD Manager, bisogna fare click destro su un AVD e selezionare **Duplicate**, o, in alternativa, selezionare Menu ▼ e poi su **Duplicate**. Apparirà la pagina **Verify Configuration**.
- Cliccare, poi, su **Change** o **Previous** se è necessario apportare modifiche alle pagine di **System Image** e **Select Hardware**.
- Apportare le modifiche ed, infine, cliccare su **Finish.**
  Adesso, l'AVD viene visualizzato nella pagina **Your Virtual Devices.**

Ritorniamo alla prima parte e esaminiamo il codice generato dalle asserzioni che abbiamo imposto.



```
@LargeTest
@RunWith(AndroidJUnit4.class)
public class MainActivityTest {

    @Rule
    public ActivityTestRule<MainActivity> mActivityTestRule = new ActivityTestRule<>(MainActivity.class);

    @Test
    public void mainActivityTest() {
        ViewInteraction textView = onView(
                allOf(withText("Hello World!"),
                        childAtPosition(
                                childAtPosition(
                                        withId(android.R.id.content),
                                        position: 0),
                                position: 0),
                        isDisplayed()));
        textView.check(matches(withText("Hello World!")));

        ViewInteraction textView2 = onView(
                allOf(withText("Hello World!"),
                        childAtPosition(
                                childAtPosition(
                                        withId(android.R.id.content),
                                        position: 0),
                                position: 0),
                        isDisplayed()));
        textView2.check(matches(isDisplayed()));

        ViewInteraction textView3 = onView(
                allOf(withText("My Application3"),
                        childAtPosition(
                                allOf(withId(R.id.action_bar),
                                        childAtPosition(
                                                withId(R.id.action_bar_container),
                                                position: 0)),
                                position: 0),
                        isDisplayed()));
        textView3.check(matches(withText("My Application3")));
    }

    private static Matcher<View> childAtPosition(
            final Matcher<View> parentMatcher, final int position) {

        return new TypeSafeMatcher<View>() {
            @Override
            public void describeTo(Description description) {
                description.appendText("Child at position " + position + " in parent ");
                parentMatcher.describeTo(description);
            }

            @Override
            public boolean matchesSafely(View view) {
                ViewParent parent = view.getParent();
                return parent instanceof ViewGroup && parentMatcher.matches(parent)
                        && view.equals(((ViewGroup) parent).getChildAt(position));
            }
```

Fig 2.31 Codice generato

Considerazioni: primo caso

```
ViewInteraction textView2 = onView(
        allOf(withText("Hello World!"),
                childAtPosition(
                        childAtPosition(
                                withId(android.R.id.content),
                                position: 0),
                        position: 0),
                isDisplayed()));
textView2.check(matches(isDisplayed()));
```

Fig 2.32 Primo caso

- Il campo di testo è stato riconosciuto in base a:
- Il testo visualizzato (“Hello World!”)
- L’id del suo genitore (android.R.id.content) che è corrispondente al widget ListViee
- Alla posizione del genitore all’interno del LinearLayout anonimo che lo contiene (posizione 0)
- Al fatto di essere visibile in quel momento sullo schermo

2.10 Valutazione dei tool disponibili

Obiettivo, quindi, di questo capitolo, dopo aver analizzato i singoli tool di Capture & Replay disponibili, è quello di valutare il tool di C&R che confà ai nostri criteri target. La discriminante sulla quale si è basata la nostra scelta è stata l’assenza del codice sorgente e, conseguentemente, la presenza del solo APK. In particolare, i Visual Tool esaminati, Sikuli e EyeAutomate, sebbene di semplice utilizzo e open source, non consentono la rieseguibilità degli script. Discorso analogo vale per i tool ODBR e RERAN. Espresso Test Recorder e RANDR (quest’ultimo, attualmente non disponibile), invece, richiedono la presenza del codice sorgente. Motivo per il quale, tali tool sono stati esclusi dalla nostra scelta. ROBOTIUM, invece, risulta essere disponibili solo per vecchie versioni OS di Android e non può essere considerato un tool open source dal momento che solo i primi 10 test sono gratuiti. Infine, SELENDROID sebbene possa risultare un tool valido, attualmente è in disuso. Dunque, avvalendoci anche della tabella 2.23, che riporta i tool esaminati in funzione dei criteri target di nostro interesse, la scelta del tool di C&R, sulla quale proseguirà la nostra analisi nei capitoli successivi, ricade su Appium.

Tabella1 Schema di classificazione finale dei tool valutati

	Tool	Funzionalità
1	Sikuli	Rientra nella categoria dei Visual tool. Semplicità di utilizzo. Viene preferito quando gli elementi dell'interfaccia utente sono stabili e non cambiano frequentemente.
2	EyeAutomate	È un Visual Tool. Rappresenta l'evoluzione di Sikuli. Esecuzione dei test veloce. Semplifica l'esecuzione dei test di regressione.
3	RANDR	Tool di Capture and Replay. È in grado di riprodurre il comportamento delle applicazioni Android su più dispositivi. Riproduce le interazioni dell'interfaccia utente in maniera indipendente dalle coordinate.
4	ODBR	Tool di supporto per la segnalazione dei bug.
5	RERAN	Tool di Capture and Replay. Riproduce gli eventi di basso livello che vengono triggerati sul dispositivo. Potente strumento di debugging.
6	Appium	Tool di automazione di test per applicazioni native, ibride e web su piattaforma Android, iOS e Windows. Tool multipiattaforma. Evoluzione di Selendroid.
7	Robotium	Framework di automazione di test case per applicazioni native ed ibride su piattaforma Android. Non aggiornato.
8	Selendroid	Tool per l'automazione dei test. È il corrispettivo di Selenium per applicazioni Android. Non aggiornato.

Commentato [PT3]: Aggiungi un miniparagrafo a questo punto del testo per spiegare che tutte le considerazioni fatte finora sono state riassunte nelle due tabelle che seguono e che in base a queste considerazioni (sono nella seconda tabella ma puoi ripeterle discorsivamente) si è scelto di proseguire l'analisi in maniera sperimentale su Appium, come verrà mostrato nei capitoli successivi

	Tool	Tecnologia Target	Disponibilità	Costo	APK/source code	Script Rieseguibili
1	Sikuli	Visual Tool	✓	Open Source	APK	×
2	EyeAutomate	Visual Tool	✓	Open Source	APK	×
3	RANDR	C&R Tool	×	A pagamento	Source code	✓
4	ODBR	Tool di debugging	×	Open Source	APK	×
5	RERAN	C&R Tool	×	Open Source	APK	×
6	APPIUM	C&R Tool	✓	Open Source	APK	✓
7	ROBOTIUM	C&R Tool	Disponile per vecchie versioni di Android	primi 10 test sono gratui	APK	✓
8	SELENDROID	C&R Tool	In disuso	Open Source	APK	✓
9	Espresso Test Recorder	C&R Tool	✓	Open Source	Source code	✓

Tabella 2.33 Classificazione dei tool esaminati in base ai criteri target di nostro interesse.

3. Analisi sperimentale

Commentato [PT4]: Aggiungi qualche altra frase che spieghi le motivazioni di questo esperimento, cioè valutare sperimentalmente la validità dello strumento Appium per testing C&R di applicazioni Android, i suoi limiti e le sue potenzialità.

Dopo aver effettuato l'analisi e la valutazione dei tool di Capture & Replay attualmente in uso, esaminando per ciascuno di essi le caratteristiche e lo scenario di utilizzo, è stato scelto Appium come tool con il quale continuare la nostra analisi sperimentale, in quanto conforme ai criteri target di nostro interesse. Obiettivo della nostra analisi è quello di valutare, in toto, la validità del tool Appium per il testing funzionale black box di applicazioni Android [13], e, di conseguenza, i suoi limiti e le sue potenzialità. In particolare, in questo capitolo ci focalizzeremo sulla fase di registrazione (Capture) di alcuni casi di test, mediante l'utilizzo del tool open source Appium, e sulla fase di riproduzione (Replay) dei medesimi mediante l'ausilio dell'IDE Eclipse. Le app oggetto della nostra analisi saranno: Munchlife, Trolly, DIAB e un'applicazione per veicoli commerciali. Le versioni dei tool utilizzate, al momento della stesura della tesi, sono le seguenti: Appium 1.17.1 ed Eclipse 2020-06. Dapprima, cominciamo con la descrizione della fase di Capture per poi proseguire con quella di Replay.

3.1 Fase di Capture

Nella fase di Capture, una volta lanciato il tool Appium, bisogna premere su *Start Server* e, successivamente, su *New Session Window*, per cominciare una nuova sessione. Come già detto in precedenza, essenziale per poter iniziare una sessione di registrazione è la definizione delle *Desired Capabilities*, le quali vengono inviate durante l'inizializzazione della sessione. Ad esempio, nel nostro primo caso in esame, viene utilizzato un emulatore WGA (Nexus S), con API 26, corrispondente alla versione Android Oreo 8.0. Il JSON delle Desired Capabilities, corrispondente al nostro caso in esame, è il seguente:

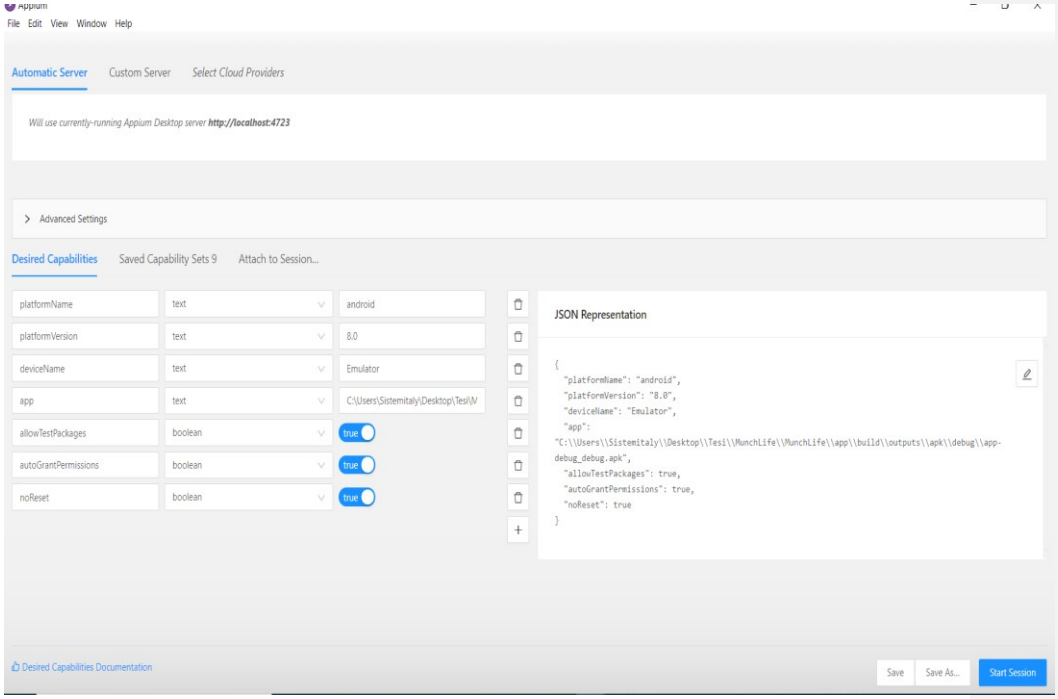


Fig. 3.1 Desired Capabilities del primo caso d'esempio.

Come si può facilmente intuire dalla Fig. 3.1, gli attributi definiti nel file JSON sono quelli già spiegati nella descrizione del tool Appium. In aggiunta all'insieme minimo di funzionalità richieste, abbiamo inserito:

- *autoGrantPermission*: Chiede ad Appium di determinare automaticamente quali autorizzazioni sono richieste dall'app e concederle all'app durante l'installazione. Il valore predefinito è settato su *false*. Nel nostro caso, lo abbiamo settato su *true*. Tuttavia, questa capability non funziona se *noReset* è settato a *true*.
- *allowTestPackages*: Consente ad Appium di installare pacchetti di Test. Nel nostro caso, tale capability è stata settata a *true*.

Una volta definite le *Desired Capabilities*, è possibile cominciare la fase di record vera e propria, premendo su *Start Session*. La prima app sulla quale andremo a registrare e rieseguire i nostri test sarà Munchlife.

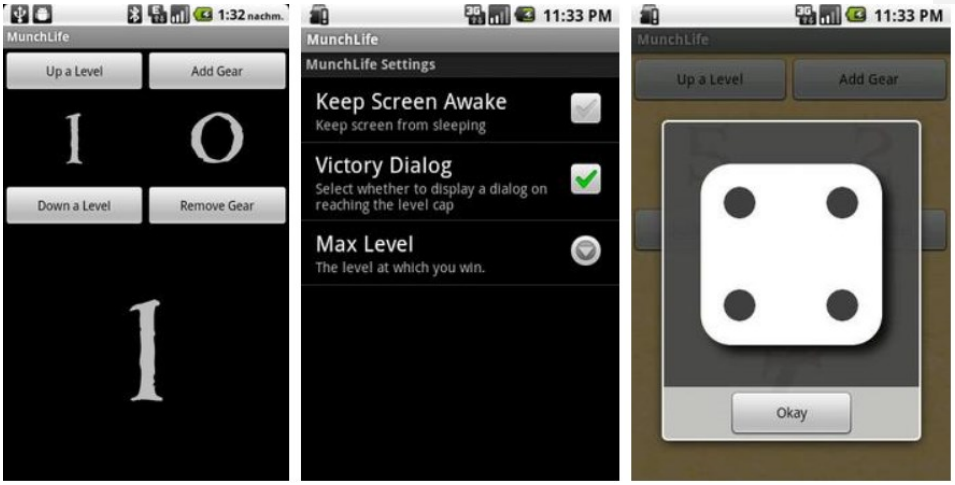


Fig. 3.2 Munchlife

Munchlife è un'applicazione molto semplice. Si tratta di un'app contatore che consente di tenere traccia del livello del personaggio mentre si gioca a carte. L'ultima release risale al 2015, di conseguenza, si tratta di un'app piuttosto obsoleta. Il primo test case previsto è registrato su un WVGA (Nexus S) con API 26, in modalità portrait.

Di seguito, vengono riportate le azioni effettuate:

- premere due volte, in successione, sul bottone UP A LEVEL;
- premere tre volte, in successione, sul bottone ADD GEAR;
- premere due volte, in successione, il bottone DOWN A LEVEL;
- premere tre volte, in successione, il bottone REMOVE GEAR;

Queste azioni, sono così registrate nei log:

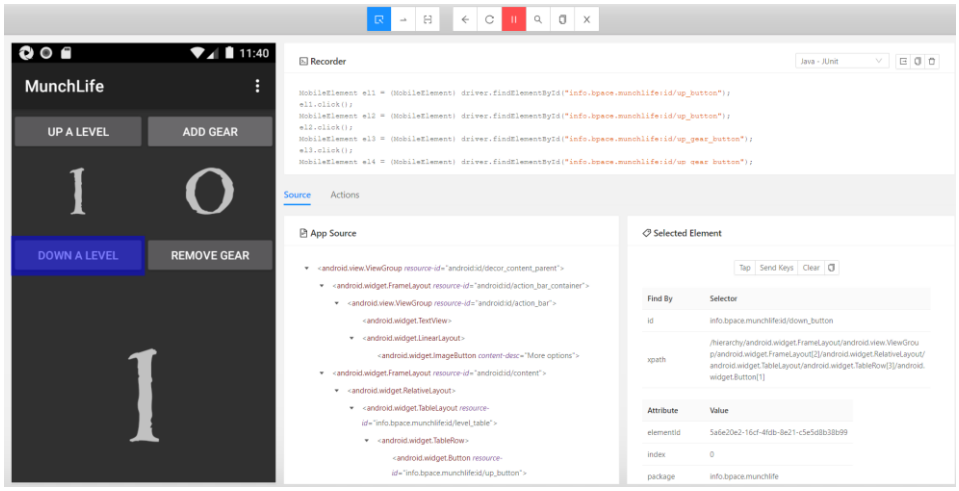


Fig. 3.3 a) a sinistra, interazione con l'emulatore; b) in alto, nella finestra Recorder, la registrazione delle azioni.

Come si può notare dalla Figura 3.3, l'Inspector di Appium consente di registrare e riprodurre il comportamento dell'applicazione nativa ispezionando il DOM e riprodurre lo script di test in qualsiasi linguaggio desiderato. Nel nostro caso, abbiamo scelto che lo script di test debba essere scritto in Java-JUnit. Per avviare la registrazione, bisogna premere sul bottone *Start Recording*, facilmente identificabile. Al termine della registrazione, è possibile premere sul bottone PAUSE RECORDING per mettere in pausa la registrazione. Adesso, non bisogna far altro che esportare il nostro script all'interno dell'IDE Eclipse per poterlo rieseguire. Di seguito, è mostrato il codice generato, corrispondente alle azioni precedentemente descritte:

```
1.  @Test
2.    public void sampleTest() {
3.      MobileElement el1 = (MobileElement) driver.findElementById("info.bpace.munchlife:id/up
    _button");
4.      el1.click();
5.      el1.click();
6.      MobileElement el2 = (MobileElement) driver.findElementById("info.bpace.munchlife:id/up
    _gear_button");
7.      el2.click();
8.      el2.click();
9.      el2.click();
10.     MobileElement el3 = (MobileElement) driver.findElementById("info.bpace.munchlife:id/do
    wn_button");
11.     el3.click();
```

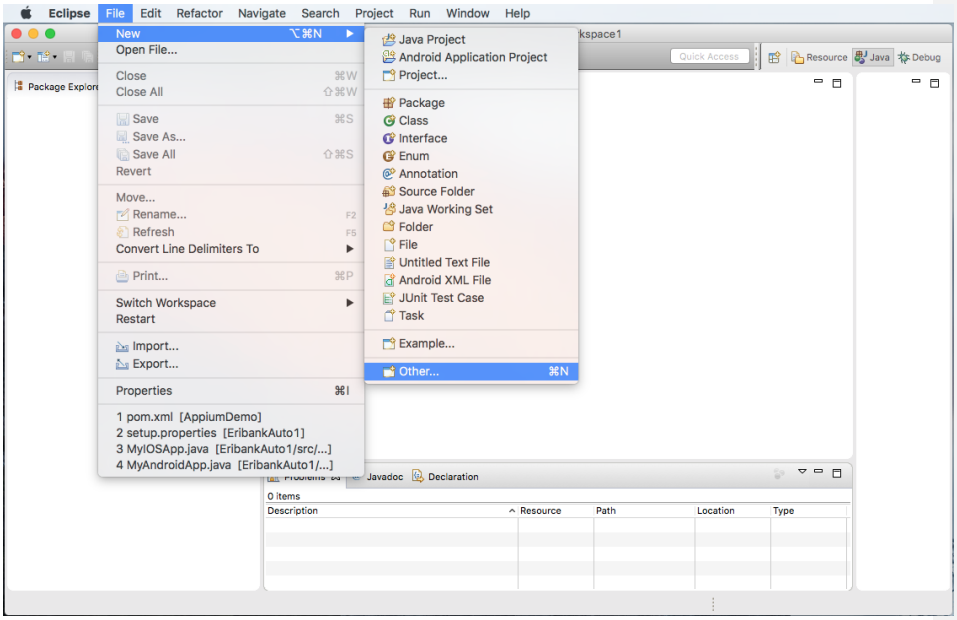
```
12.     el3.click();
13.     MobileElement el4 = (MobileElement) driver.findElementById("info.bpace.munchlife:id/do
    wn_gear_button");
14.     el4.click();
15.     el4.click();
16.     el4.click();
17.   }
```

Fig. 3.4 Individuazione e click dei pulsanti UP A LEVEL, DOWN A LEVEL, ADD GEAR, REMOVE GEAR.

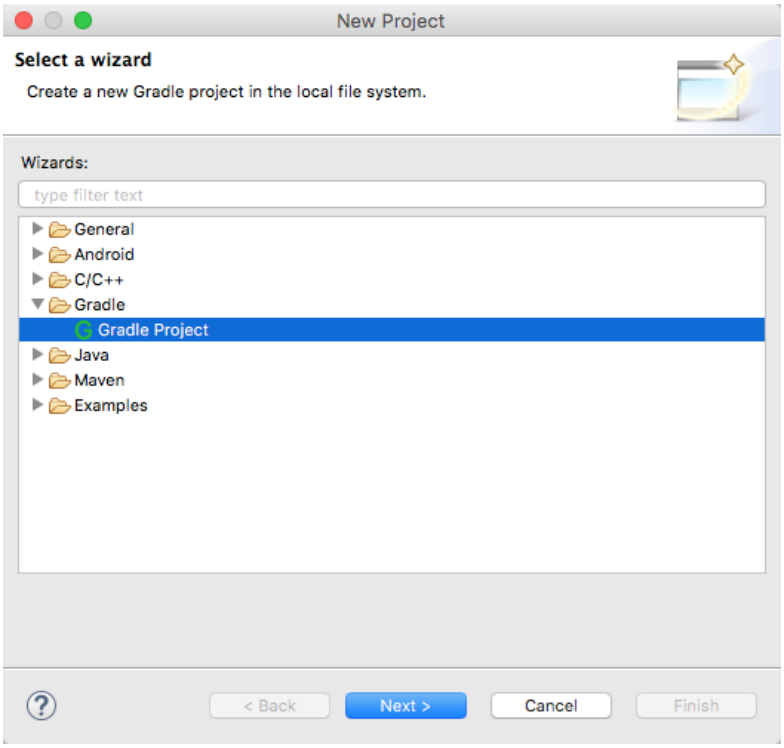
3.2 Fase di Replay

Durante la fase di Replay viene eseguito il test case come era stato precedentemente registrato su Appium [18]. In particolare, in Eclipse, bisogna:

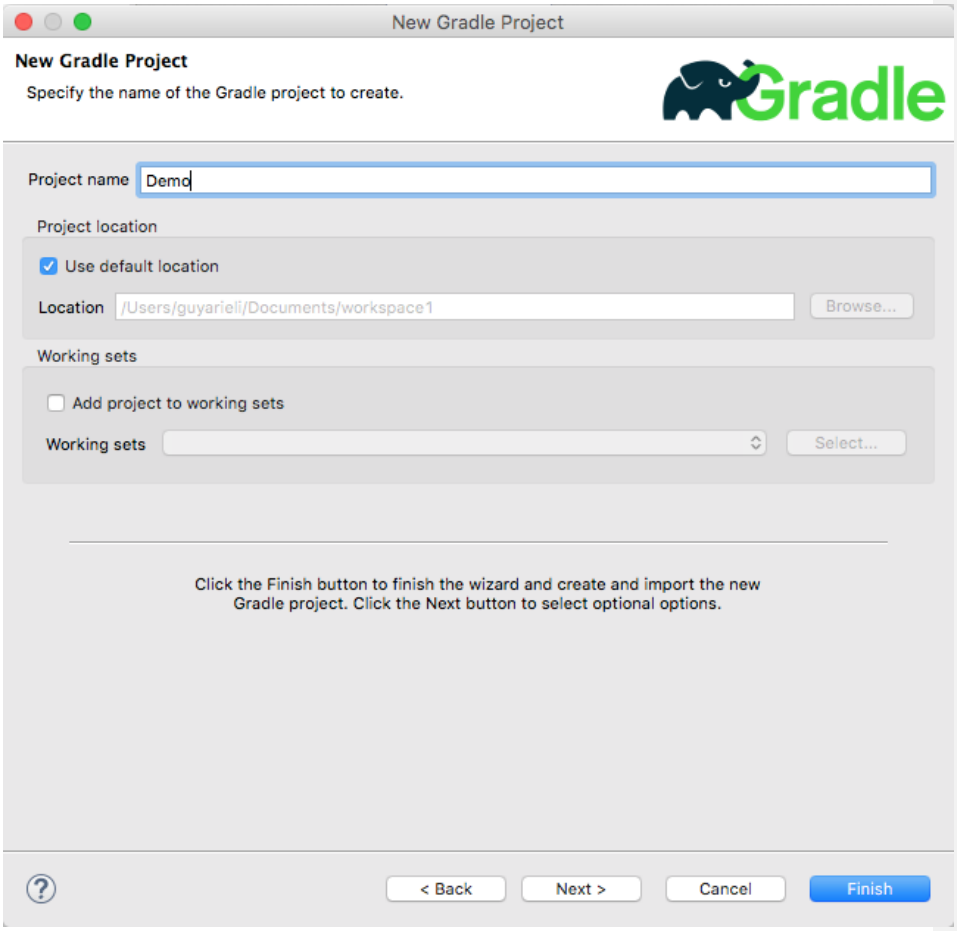
1. creare un new Gradle project, **File->New->Other...**



2. Selezionare ‘Gradle Project’



3. Premere su ‘Create a simple project (skip archetype selection)’, quindi, cliccare su ‘Next’. Un’attenzione particolare deve essere posta sul nome del progetto. Eclipse richiede che il nome del progetto Gradle cominci con una lettera minuscola anche se ciò non viene specificato in alcun modo. Tuttavia, nel caso in cui si crei un progetto con lettera minuscola, l’esecuzione del test fallisce.



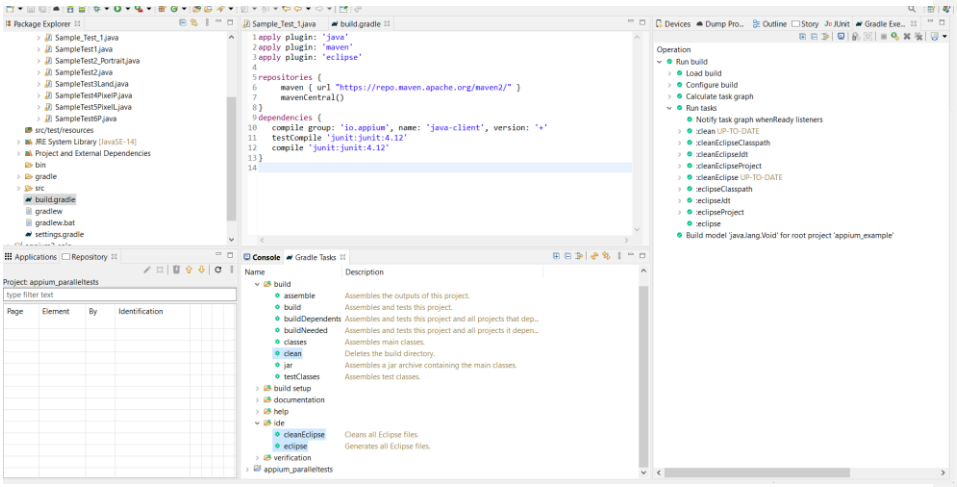
4. A questo punto, bisogna aprire il progetto appena creato e, successivamente, aprire il file build.gradle che è stato appena creato. In particolare, bisogna andare a sostituire il contenuto generale del file build.gradle con il seguente:

Sample_Test_1.java build.gradle

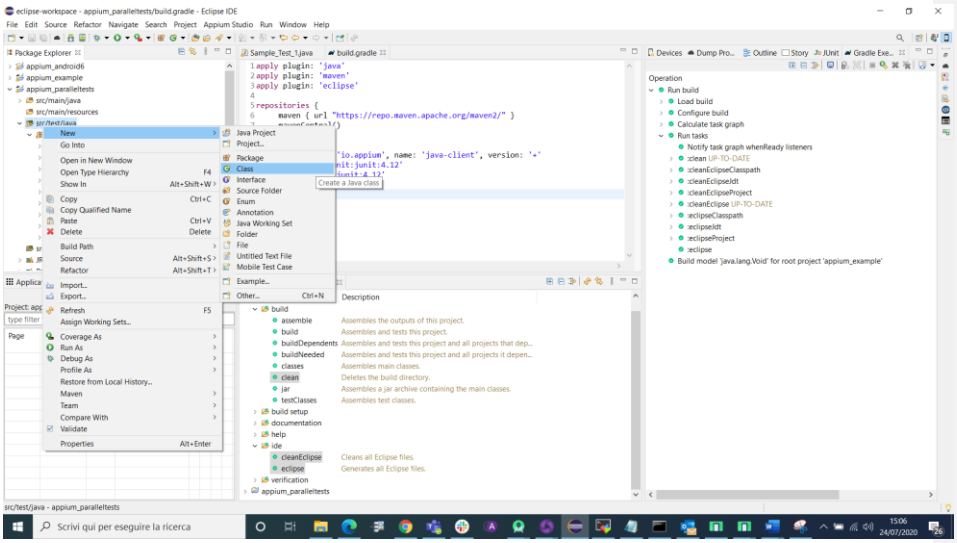
```
apply plugin: 'java'
apply plugin: 'maven'
apply plugin: 'eclipse'

repositories {
    maven { url "https://repo.maven.apache.org/maven2/" }
    mavenCentral()
}
dependencies {
  compile group: 'io.appium', name: 'java-client', version: '+'
  testCompile 'junit:junit:4.12'
  compile 'junit:junit:4.12'
}
```

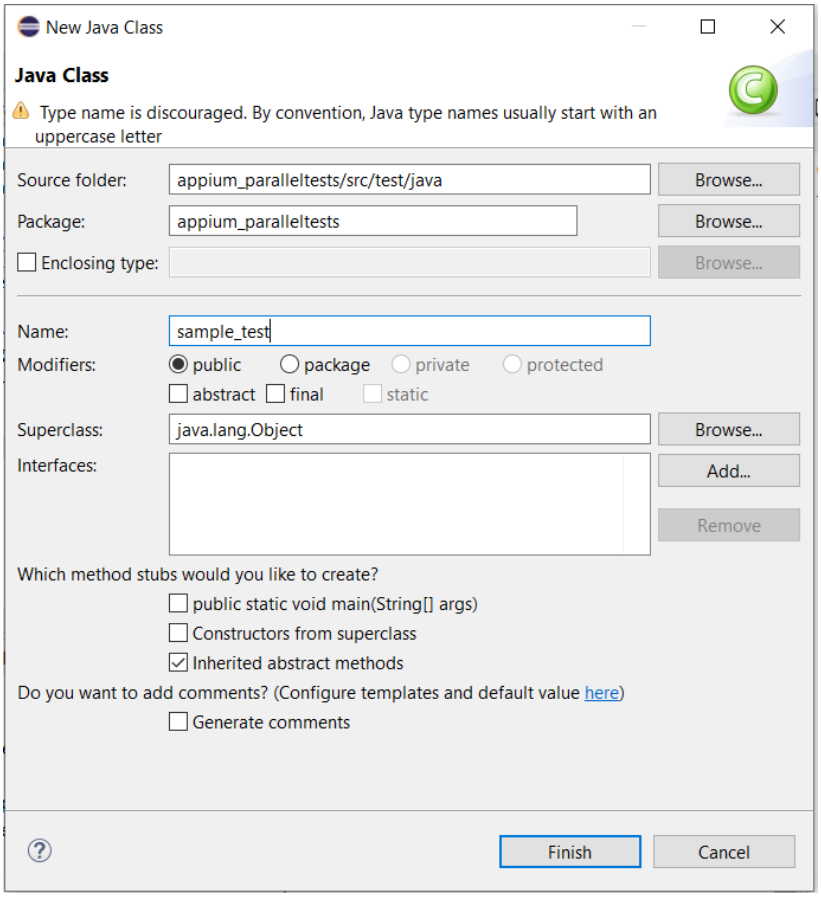
5. Poi, bisogna eseguire le attività *clean*, *cleanEclipse* ed *eclipse* per preparare l'ambiente Eclipse [19].



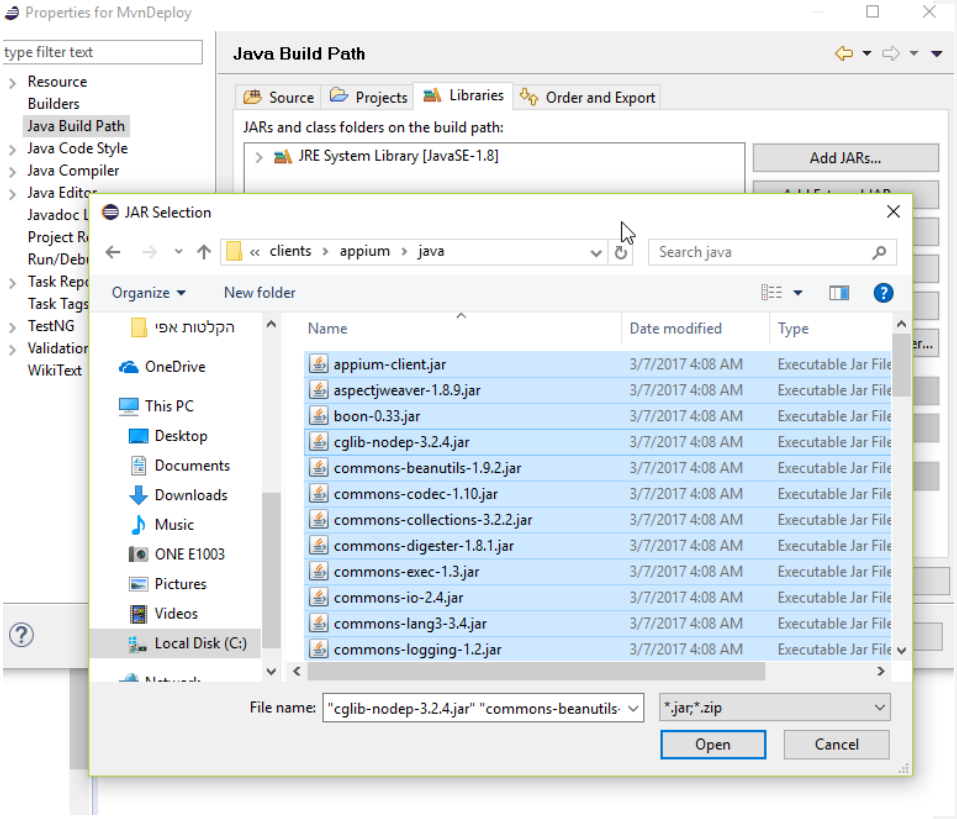
A questo punto, per creare una nuova classe di test, sarà sufficiente fare click con il tasto destro sulla cartella di origine del test, quindi, selezionare **New->Class:**



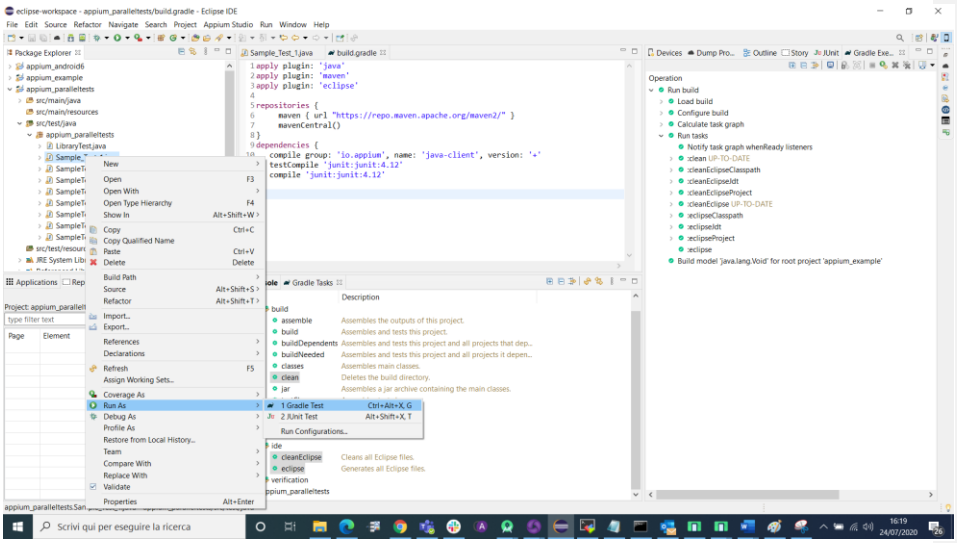
È necessario, una volta creata la classe di test, fornire un nome. È importante sottolineare come il nome della classe di test deve cominciare, stavolta, con una lettera maiuscola. Tuttavia, nonostante questo avvertimento, il tool consente di creare una classe di test anche nel caso in cui il nome di quest'ultimo non cominci con una lettera maiuscola. In tal caso, però, al momento del lancio della classe di test, si riscontreranno problemi, in quanto il test fallirà. Una volta creata la classe di test, con il nome opportuno, bisogna cliccare su 'Finish'.



A questo punto, bisogna importare il codice generato in Appium, cliccando sull'icona 'Copy to Clipboard' , ed incollarlo all'interno della classe Java appena creata. Fondamentale poi aggiungere le librerie di Appium al progetto creato. Basterà importare tutti i file JAR da <Appium Studio Installation Folder>\clients\appium\java*.jar ed aggiungerlo al JAVA build path, in particolare al Classpath e non al Modulepath, come mostrato nella seguente figura:



A questo punto, non ci resta altro che eseguire il test in Eclipse [20], facendo clic destro sulla classe di test che si intende eseguire e, quindi, cliccando su **Ras As → Gradle Test:**



In questo modo, è possibile eseguire tutte le classi di test che si vogliono includere nel progetto. Altresì importante è la funzione *Clean* accessibile dal toolbar. Essa consente di rimuovere i file già compilati nel progetto in modo da poter eseguire una rebuild completa. Nel caso in cui si intendano eseguire classi di test in successione senza aver utilizzato il comando *Clean*, ci si potrebbe trovare di fronte ad un fallimento del test legato al tool Eclipse.

Ritornando al nostro caso, il test case registrato su un WVGA (Nexus S) con API 26, in modalità portrait, in Appium, viene successivamente rieseguito sullo stesso vendor, in Eclipse. Le azioni che coinvolgono il click dei pulsanti vengono interpretate correttamente. A partire da tale emulatore, poi, sono stati provati i replay su altri dispositivi. Di seguito, gli emulatori utilizzati con una breve descrizione dell'esito del test:

- WVGA (Nexus S) API 26, stesso emulatore, modalità portrait: il replay viene eseguito correttamente
- WVGA (Nexus S) API 26, stesso emulatore, modalità landscape: il replay viene eseguito correttamente
- WVGA (Nexus S) API 27, modalità portrait, stesso emulatore ma con versione successiva del sistema operativo: il replay avviene correttamente
- WVGA (Nexus S) API 23, modalità portrait, stesso emulatore ma con versione meno aggiornata del sistema operativo: il replay avviene correttamente
- WVGA (Nexus S) API 23, modalità landscape, stesso emulatore ma con versione meno aggiornata del sistema operativo: il replay avviene correttamente

- Nexus One API 23, modalità portrait, device diverso (smartphone): il replay avviene correttamente.
- Nexus 10 API 23, modalità portrait, device diverso (Tablet): il replay avviene correttamente
- Nexus 10 API 23, modalità landscape, device diverso (Tablet): il replay avviene correttamente
- Nexus 10 API 26, modalità portrait, device diverso (Tablet): il replay avviene correttamente
- Nexus 10 API 26, modalità landscape, device diverso (Tablet): il replay avviene correttamente.

Di seguito, una tabella riassuntiva del primo scenario di test.

ID	Descrizione	Caratteristiche Emulatore	Modalità	Versione di Android	Input	Esito
Test Case1_1	Funzionamento dei pulsanti per aggiungere e ridurre GEAR e LEVEL, volto a valutare la corretta gestione del valor nullo.	WVGA (Nexus S) - Smartphone	Portrait	API 26	1. Premere due volte in successione UP A LEVEL 2. Premere tre volte in successione ADD GEAR 3. Premere due volte in successione DOWN A LEVEL 4. Premere tre volte in successione REMOVE GEAR.	Il Replay coincide con la fase di Capture
Test Case1_2	Funzionamento dei pulsanti per aggiungere e ridurre GEAR e LEVEL, volto a valutare la corretta gestione del valor nullo.	WVGA (Nexus S) - Smartphone	Landscape	API 26	1. Premere due volte in successione UP A LEVEL 2. Premere tre volte in successione ADD GEAR 3. Premere due volte in successione DOWN A LEVEL 4. Premere tre volte in successione REMOVE GEAR.	Il Replay coincide con la fase di Capture
Test Case1_3	Funzionamento dei pulsanti per aggiungere e ridurre GEAR e LEVEL, volto a valutare la corretta gestione del valor nullo.	WVGA (Nexus S) - Smartphone	Portrait	API 27	1. Premere due volte in successione UP A LEVEL 2. Premere tre volte in successione ADD GEAR 3. Premere due volte in successione DOWN A LEVEL 4. Premere tre volte in successione REMOVE GEAR.	Il Replay coincide con la fase di Capture
Test Case1_4	Funzionamento dei pulsanti per aggiungere e ridurre GEAR e LEVEL, volto a valutare la corretta gestione del valor nullo.	WVGA (Nexus S) - Smartphone	Portrait	API 23	1. Premere due volte in successione UP A LEVEL 2. Premere tre volte in successione ADD GEAR 3. Premere due volte in successione DOWN A LEVEL 4. Premere tre volte in successione REMOVE GEAR.	Il Replay coincide con la fase di Capture
Test Case1_5	Funzionamento dei pulsanti per aggiungere e ridurre GEAR e LEVEL, volto a valutare la corretta gestione del valor nullo.	WVGA (Nexus S) - Smartphone	Landscape	API 24	1. Premere due volte in successione UP A LEVEL 2. Premere tre volte in successione ADD GEAR 3. Premere due volte in successione DOWN A LEVEL 4. Premere tre volte in successione REMOVE GEAR.	Il Replay coincide con la fase di Capture
Test Case1_6	Funzionamento dei pulsanti per aggiungere e ridurre GEAR e LEVEL, volto a valutare la corretta gestione del valor nullo.	Nexus One - Smartphone	Portrait	API 23	1. Premere due volte in successione UP A LEVEL 2. Premere tre volte in successione ADD GEAR 3. Premere due volte in successione DOWN A LEVEL 4. Premere tre volte in successione REMOVE GEAR.	Il Replay coincide con la fase di Capture
Test Case1_7	Funzionamento dei pulsanti per aggiungere e ridurre GEAR e LEVEL, volto a valutare la corretta gestione del valor nullo.	Nexus 10 - Tablet	Portrait	API 23	1. Premere due volte in successione UP A LEVEL 2. Premere tre volte in successione ADD GEAR 3. Premere due volte in successione DOWN A LEVEL 4. Premere tre volte in successione REMOVE GEAR.	Il Replay coincide con la fase di Capture
Test Case1_8	Funzionamento dei pulsanti per aggiungere e ridurre GEAR e LEVEL, volto a valutare la corretta gestione del valor nullo.	Nexus 10 - Tablet	Landscape	API 23	1. Premere due volte in successione UP A LEVEL 2. Premere tre volte in successione ADD GEAR 3. Premere due volte in successione DOWN A LEVEL 4. Premere tre volte in successione REMOVE GEAR.	Il Replay coincide con la fase di Capture
Test Case1_9	Funzionamento dei pulsanti per aggiungere e ridurre GEAR e LEVEL, volto a valutare la corretta gestione del valor nullo.	Nexus 10 - Tablet	Portrait	API 26	1. Premere due volte in successione UP A LEVEL 2. Premere tre volte in successione ADD GEAR 3. Premere due volte in successione DOWN A LEVEL 4. Premere tre volte in successione REMOVE GEAR.	Il Replay coincide con la fase di Capture
Test Case1_10	Funzionamento dei pulsanti per aggiungere e ridurre GEAR e LEVEL, volto a valutare la corretta gestione del valor nullo.	Nexus 10 - Tablet	Landscape	API 26	1. Premere due volte in successione UP A LEVEL 2. Premere tre volte in successione ADD GEAR 3. Premere due volte in successione DOWN A LEVEL 4. Premere tre volte in successione REMOVE GEAR.	Il Replay coincide con la fase di Capture

Commentato [PT5]: Metti sempre le didascalie a tabelle/figure

Fig. 3.5 Test Case 1

Il secondo scenario di test che prevediamo consiste nel verificare la possibilità di premere in successione sul bottone UP A LEVEL, due volte in successione sul bottone ADD GEAR, il lancio combinatorio del dado e, infine, il reset. Il test è stato registrato su un WVGA (Nexus S) con API 26, in modalità portrait. Le azioni da eseguire solo le seguenti:

1. Premere tre volte in successione il bottone UP A LEVEL;
2. Premere due volte in successione il bottone ADD GEAR;
3. Premere sui tre puntini corrispondenti al menù laterale – More Options;
4. Premere su ROLL DICE;
5. Premere su Okay;
6. Premere sui tre puntini corrispondenti al menù laterale – More Options;
7. Premere su RESET;

Di seguito, la porzione di codice corrispondente agli step del Test Case 2 appena descritto:

```
1.  @Test
2.   public void sampleTest() throws InterruptedException{
3.     MobileElement el1 = (MobileElement) driver.findElementById("info.bpace.munchlife:id/up_
    button");
4.     el1.click();
5.     el1.click();
6.     el1.click();
7.     MobileElement el2 = (MobileElement) driver.findElementById("info.bpace.munchlife:id/up_
    gear_button");
8.     el2.click();
9.     el2.click();
10.    MobileElement el3 = (MobileElement) driver.findElementByAccessibilityId("More options")
    ;
11.    el3.click();
12.    Thread.sleep(2000);
13.    MobileElement el4 = (MobileElement) driver.findElementByXPath("/hierarchy/android.widge
    t.FrameLayout/android.widget.FrameLayout/android.widget.ListView/android.widget.LinearLayo
    ut[2]/android.widget.RelativeLayout/android.widget.TextView");
14.    el4.click();
15.    MobileElement el5 = (MobileElement) driver.findElementById("android:id/button3");
16.    el5.click();
17.    MobileElement el6 = (MobileElement) driver.findElementByAccessibilityId("More options")
    ;
18.    el6.click();
19.    Thread.sleep(2000);
20.    MobileElement el7 = (MobileElement) driver.findElementByXPath("/hierarchy/android.widge
    t.FrameLayout/android.widget.FrameLayout/android.widget.ListView/android.widget.LinearLayo
    ut[1]/android.widget.RelativeLayout/android.widget.TextView");
21.    el7.click();
22.  }
```

Fig. 3.6 Codice corrispondente alle azioni previste dal Test Case 2.

Il test è stato, poi, rieseguito in Eclipse sui seguenti dispositivi con descrizione dell'esito del test:

- WVGA (Nexus S) con API 26, modalità portrait (stesso device): il replay viene eseguito correttamente;
- WVGA (Nexus S) con API 26, modalità landscape (stesso device): il replay viene eseguito correttamente;
- WVGA (Nexus S) con API 23, modalità portrait (stesso device): il replay viene eseguito correttamente;
- WVGA (Nexus S) con API 23, modalità landscape (stesso device): il replay viene eseguito correttamente;
- Tablet Nexus 10 con API 27, modalità portrait (device differente): il replay viene eseguito correttamente;
- Tablet Nexus 10 con API 27, modalità landscape (device differente): il replay viene eseguito correttamente;

ID	Descrizione	Caratteristiche Emulatore	Modalità	Versione di Android	Input	Esito
Test Case2_1	Funzionamento dei pulsanti per aggiungere e ridurre GEAR e LEVEL, menù laterale, lancio dei dadi e reset.	WVGA (Nexus S) - Smartphone	Portrait	API 26	1. Premere tre volte in successione UP A LEVEL; 2. Premere due volte in successione ADD GEAR; 3. Premere sul menù laterale - MORE OPTIONS; 4. Premere su ROLL DICE; 5. Premere su Okay; 6. Premere sul menù laterale - MORE OPTIONS; 7. Premere su RESET.	Il Replay coincide con la fase di Capture
Test Case2_2	Funzionamento dei pulsanti per aggiungere e ridurre GEAR e LEVEL, menù laterale, lancio dei dadi e reset.	WVGA (Nexus S) - Smartphone	Landscape	API 26	1. Premere tre volte in successione UP A LEVEL; 2. Premere due volte in successione ADD GEAR; 3. Premere sul menù laterale - MORE OPTIONS; 4. Premere su ROLL DICE; 5. Premere su Okay; 6. Premere sul menù laterale - MORE OPTIONS; 7. Premere su RESET.	Il Replay coincide con la fase di Capture
Test Case2_3	Funzionamento dei pulsanti per aggiungere e ridurre GEAR e LEVEL, menù laterale, lancio dei dadi e reset.	WVGA (Nexus S) - Smartphone	Portrait	API 23	1. Premere tre volte in successione UP A LEVEL; 2. Premere due volte in successione ADD GEAR; 3. Premere sul menù laterale - MORE OPTIONS; 4. Premere su ROLL DICE; 5. Premere su Okay; 6. Premere sul menù laterale - MORE OPTIONS; 7. Premere su RESET.	Il Replay coincide con la fase di Capture
Test Case2_4	Funzionamento dei pulsanti per aggiungere e ridurre GEAR e LEVEL, menù laterale, lancio dei dadi e reset.	WVGA (Nexus S) - Smartphone	Landscape	API 23	1. Premere tre volte in successione UP A LEVEL; 2. Premere due volte in successione ADD GEAR; 3. Premere sul menù laterale - MORE OPTIONS; 4. Premere su ROLL DICE; 5. Premere su Okay; 6. Premere sul menù laterale - MORE OPTIONS; 7. Premere su RESET.	Il Replay coincide con la fase di Capture
Test Case2_5	Funzionamento dei pulsanti per aggiungere e ridurre GEAR e LEVEL, menù laterale, lancio dei dadi e reset.	Nexus 10 - Tablet	Portrait	API 27	1. Premere tre volte in successione UP A LEVEL; 2. Premere due volte in successione ADD GEAR; 3. Premere sul menù laterale - MORE OPTIONS; 4. Premere su ROLL DICE; 5. Premere su Okay; 6. Premere sul menù laterale - MORE OPTIONS; 7. Premere su RESET.	Il Replay coincide con la fase di Capture
Test Case2_6	Funzionamento dei pulsanti per aggiungere e ridurre GEAR e LEVEL, menù laterale, lancio dei dadi e reset.	Nexus 10 - Tablet	Landscape	API 27	1. Premere tre volte in successione UP A LEVEL; 2. Premere due volte in successione ADD GEAR; 3. Premere sul menù laterale - MORE OPTIONS; 4. Premere su ROLL DICE; 5. Premere su Okay; 6. Premere sul menù laterale - MORE OPTIONS; 7. Premere su RESET.	Il Replay coincide con la fase di Capture

Fig. 3.7 Test Case 2

Nel terzo scenario di test, invece, vogliamo testare la possibilità di premere due volte in successione su UP A LEVEL, premere una volta su ADD GEAR, di premere due volte in successione sul simbolo

GENDER, il lancio combinatorio e, infine, il reset. Il test è stato registrato su un WVGA (Nexus S) con API 26, stavolta in modalità landscape. Si tratta dell'equivalente opposto al Test Case precedente. Le azioni da eseguire sono le seguenti:

1. Premere due volte in successione su UP A LEVEL;
2. Premere una volta su ADD GEAR;
3. Premere due volte in successione sul simbolo GENDER;
4. Premere sui tre puntini corrispondenti al menù laterale – More Options;
5. Premere su ROLL DICE;
6. Premere su Okay;
7. Premere nuovamente sui tre puntini laterali corrispondenti al menù laterale – More Options;
8. Premere su RESET;

Di seguito, porzione del codice corrispondente agli step del terzo Test Case:

```
1.  @Test
2.   public void sampleTest() throws InterruptedException{
3.     MobileElement el1 = (MobileElement) driver.findElementById("info.bpace.munchlife:id/up_
    button");
4.     el1.click();
5.     MobileElement el2 = (MobileElement) driver.findElementById("info.bpace.munchlife:id/up_
    button");
6.     el2.click();
7.     MobileElement el3 = (MobileElement) driver.findElementById("info.bpace.munchlife:id/up_
    gear_button");
8.     el3.click();
9.     MobileElement el4 = (MobileElement) driver.findElementById("info.bpace.munchlife:id/gen
    der");
10.    el4.click();
11.    el4.click();
12.    MobileElement el5 = (MobileElement) driver.findElementByAccessibilityId("More options")
    ;
13.    el5.click();
14.    Thread.sleep(2000);
15.    MobileElement el6 = (MobileElement) driver.findElementByXPath("/hierarchy/android.widge
    t.FrameLayout/android.widget.FrameLayout/android.widget.ListView/android.widget.LinearLayo
    ut[2]/android.widget.RelativeLayout/android.widget.TextView");
16.    el6.click();
17.    MobileElement el7 = (MobileElement) driver.findElementById("android:id/button3");
18.    el7.click();
19.    MobileElement el8 = (MobileElement) driver.findElementByAccessibilityId("More options")
    ;
20.    el8.click();
21.    Thread.sleep(2000);
22.    MobileElement el9 = (MobileElement) driver.findElementByXPath("/hierarchy/android.widge
    t.FrameLayout/android.widget.FrameLayout/android.widget.ListView/android.widget.LinearLayo
    ut[1]/android.widget.RelativeLayout/android.widget.TextView");
23.    el9.click();
24.  }
```

Fig. 3.8 Codice corrispondente agli step previsti dal terzo Test Case.

Il test è stato, poi, rieseguito in Eclipse sui seguenti dispositivi con descrizione dell'esito del test:

- WVGA (Nexus S) con API 26, modalità landscape (stesso device): il replay viene eseguito correttamente;
- WVGA (Nexus S) con API 26, modalità portrait (stesso device): il replay non viene eseguito correttamente. Il test fallisce al punto 3 in quanto il sistema non trova il simbolo del GENDER;
- WVGA (Nexus S) con API 23, modalità landscape (stesso device): il replay viene eseguito correttamente;
- WVGA (Nexus S) con API 23, modalità portrait (stesso device): il replay non viene eseguito correttamente. Il test fallisce al punto 3, dualmente al WVGA (Nexus S) con API 26, in modalità sempre portrait, in quanto il sistema non trova il simbolo del GENDER;
- Tablet Nexus 10 con API 27, modalità portrait (device differente): il replay viene eseguito correttamente;
- Tablet Nexus 10 con API 27, modalità landscape (device differente): il replay viene eseguito correttamente;

A valle del test possiamo asserire che il replay fallisce nel WVGA (Nexus S) con API 26 e WVGA (Nexus S) con API 23, entrambi in modalità portrait, allo step 3 in quanto, come già detto, il sistema non riesce a trovare il simbolo del GENDER che nella modalità landscape della versione è invece presente. D'altra parte, il replay sul Tablet Nexus 10 con API 27, in modalità portrait, non fallisce in quanto il simbolo GENDER è presente come è possibile osservare dalle seguenti figure:

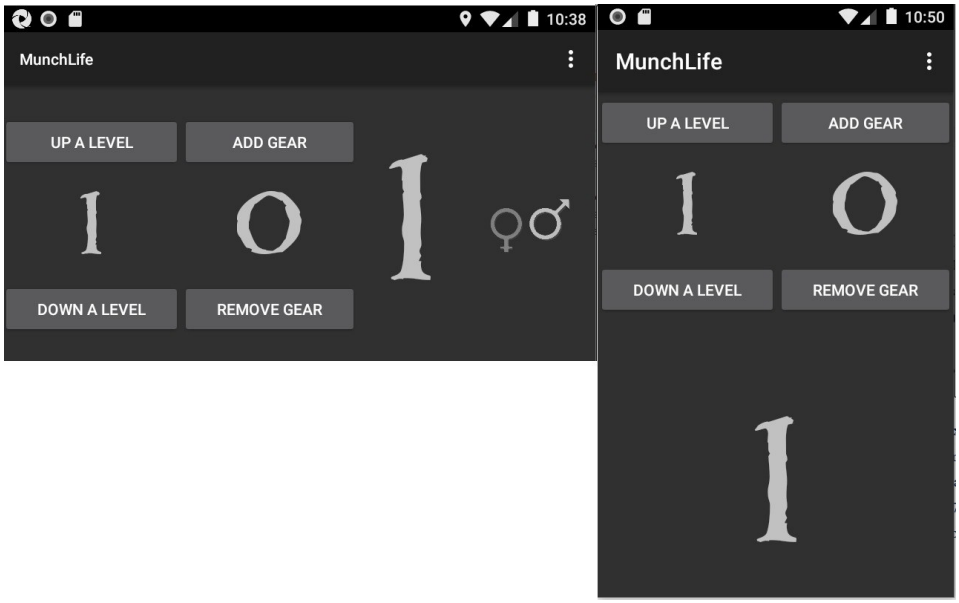


Fig. 3.9 Si può facilmente notare come il simbolo GENDER presente nella versione portrait di MunchLife su smartphone (a sinistra) non è invece presente nella versione landscape della stessa sul medesimo smartphone (a destra).

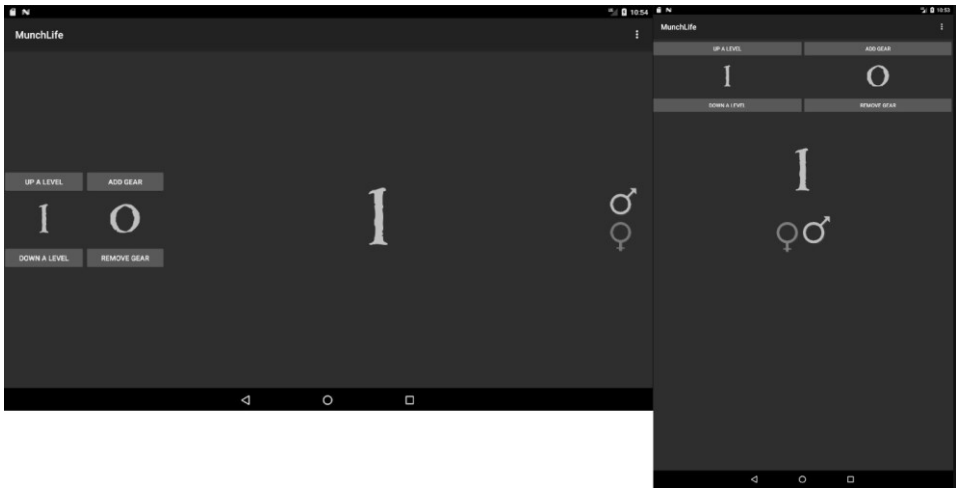


Fig. 3.10 La figura mostra come sia nella versione landscape che in quella portrait di MunchLife sul Tablet il simbolo GENDER sia sempre presente.

Verosimilmente, il layout portrait ha delle dimensioni ben definite, motivo per il quale su uno schermo piccolo, quindi non un Tablet, il simbolo GENDER finisce fuori dalla zona visibile. È possibile giungere alla conclusione, tramite l'ausilio del log, che l'altezza di alcuni campi, quali quelli con i numeri, è fissata in termini della dimensione dei caratteri del font che, conseguentemente, non riesce a diminuire al di sotto di un certo valore, consentendo al simbolo GENDER di rimanere nella zona visibile.

ID	Descrizione	Caratteristiche Emulatore	Modalità	Versione di Android	Input	Esito
Test Case3_1	Funzionamento dei pulsanti per aggiungere e ridurre GEAR e LEVEL, GENDER, lancio combinatorio e RESET.	WVGA (Nexus S) - Smartphone	Landscape	API 26	1. Premere due volte in successione UP A LEVEL; 2. Premere ADD GEAR; 3. Premere due volte in successione sul simbolo GENDER; 4. Premere sul menù laterale - MORE OPTIONS; 5. Premere ROLL DICE; 6. Premere Okay; 7. Premere sul menù laterale - MORE OPTIONS; 8. Premere RESET.	Il replay coincide con la fase di Capture
Test Case3_2	Funzionamento dei pulsanti per aggiungere e ridurre GEAR e LEVEL, GENDER, lancio combinatorio e RESET.	WVGA (Nexus S) - Smartphone	Portrait	API 26	1. Premere due volte in successione UP A LEVEL; 2. Premere ADD GEAR; 3. Premere due volte in successione sul simbolo GENDER; 4. Premere sul menù laterale - MORE OPTIONS; 5. Premere ROLL DICE; 6. Premere Okay; 7. Premere sul menù laterale - MORE OPTIONS; 8. Premere RESET.	Non viene identificato il simbolo GENDER
Test Case3_3	Funzionamento dei pulsanti per aggiungere e ridurre GEAR e LEVEL, GENDER, lancio combinatorio e RESET.	WVGA (Nexus S) - Smartphone	Landscape	API 23	1. Premere due volte in successione UP A LEVEL; 2. Premere ADD GEAR; 3. Premere due volte in successione sul simbolo GENDER; 4. Premere sul menù laterale - MORE OPTIONS; 5. Premere ROLL DICE; 6. Premere Okay; 7. Premere sul menù laterale - MORE OPTIONS; 8. Premere RESET.	Il replay coincide con la fase di Capture
Test Case3_4	Funzionamento dei pulsanti per aggiungere e ridurre GEAR e LEVEL, GENDER, lancio combinatorio e RESET.	WVGA (Nexus S) - Smartphone	Portrait	API 23	1. Premere due volte in successione UP A LEVEL; 2. Premere ADD GEAR; 3. Premere due volte in successione sul simbolo GENDER; 4. Premere sul menù laterale - MORE OPTIONS; 5. Premere ROLL DICE; 6. Premere Okay; 7. Premere sul menù laterale - MORE OPTIONS; 8. Premere RESET.	Non viene identificato il simbolo GENDER
Test Case3_5	Funzionamento dei pulsanti per aggiungere e ridurre GEAR e LEVEL, GENDER, lancio combinatorio e RESET.	Nexus 10 - Tablet	Portrait	API 27	1. Premere due volte in successione UP A LEVEL; 2. Premere ADD GEAR; 3. Premere due volte in successione sul simbolo GENDER; 4. Premere sul menù laterale - MORE OPTIONS; 5. Premere ROLL DICE; 6. Premere Okay; 7. Premere sul menù laterale - MORE OPTIONS; 8. Premere RESET.	Il replay coincide con la fase di Capture
Test Case3_6	Funzionamento dei pulsanti per aggiungere e ridurre GEAR e LEVEL, GENDER, lancio combinatorio e RESET.	Nexus 10 - Tablet	Landscape	API 27	1. Premere due volte in successione UP A LEVEL; 2. Premere ADD GEAR; 3. Premere due volte in successione sul simbolo GENDER; 4. Premere sul menù laterale - MORE OPTIONS; 5. Premere ROLL DICE; 6. Premere Okay; 7. Premere sul menù laterale - MORE OPTIONS; 8. Premere RESET.	Il replay coincide con la fase di Capture

Fig. 3.11 Test Case 3

Il quarto scenario di utilizzo che prevediamo consiste nel registrare un test su di un dispositivo in lingua inglese e, verificare la sua replicabilità, rieseguendo il medesimo test su un dispositivo in lingua italiana e spagnola. Dunque, in primis, è necessario settare la lingua di interesse, nel nostro caso l'inglese, dai system settings del device. In particolare, il test è stato registrato su un WVGA (Nexus S) con API 26, in modalità portrait. Gli step da eseguire sono i seguenti:

1. Premere tre volte in successione UP A LEVEL;
2. Premere una volta ADD GEAR;

3. Premere una volta REMOVE GEAR;
4. Premere tre volte in successione DOWN A LEVEL;
5. Premere sui tre puntini corrispondenti al menù laterale – MORE OPTIONS;
6. Premere ROLL DICE;
7. Premere OKAY;

Di seguito, porzione del codice corrispondente agli step del quarto test case:

```
1.  @Test
2.    public void sampleTest() {
3.      MobileElement el1 = (MobileElement) driver.findElementById("info.bpace.munchlife:id/up
    _button");
4.      el1.click();
5.      el1.click();
6.      el1.click();
7.      MobileElement el2 = (MobileElement) driver.findElementById("info.bpace.munchlife:id/up
    _gear_button");
8.      el2.click();
9.      MobileElement el3 = (MobileElement) driver.findElementById("info.bpace.munchlife:id/do
    wn_gear_button");
10.     el3.click();
11.     MobileElement el4 = (MobileElement) driver.findElementById("info.bpace.munchlife:id/do
    wn_button");
12.     el4.click();
13.     el4.click();
14.     el4.click();
15.     MobileElement el5 = (MobileElement) driver.findElementByAccessibilityId("More options"
    );
16.     el5.click();
17.     MobileElement el6 = (MobileElement) driver.findElementByXPath("/hierarchy/android.widg
    et.FrameLayout/android.widget.FrameLayout/android.widget.ListView/android.widget.LinearLay
    out[2]/android.widget.RelativeLayout/android.widget.TextView");
18.     el6.click();
19.     MobileElement el7 = (MobileElement) driver.findElementById("android:id/button3");
20.     el7.click();
21.   }
```

Fig. 3.12 Codice corrispondente agli step previsti dal quarto Test Case.

Il test è stato, poi, rieseguito in Eclipse sui seguenti dispositivi con descrizione dell'esito del test:

- WVGA (Nexus S) con API 26, in lingua inglese, modalità portrait (stesso device): il replay viene eseguito correttamente;
- WVGA (Nexus S) con API 26, in lingua italiana, modalità portrait (stesso device): il replay viene eseguito correttamente;
- WVGA (Nexus S) con API 26, in lingua spagnola, modalità portrait (stesso device): il replay viene eseguito correttamente;

ID	Descrizione	Caratteristiche Emulatore	Modalità	Versione di Android	Input	Esito
Test Case4_1	Funzionamento dei pulsanti per aggiungere e ridurre GEAR e LEVEL, lancio combinatorio.	WVGA (Nexus S) - Smartphone	Portrait Lingua inglese	API 26	1. Premere tre volte in successione UP A LEVEL; 2. Premere ADD GEAR; 3. Premere REMOVE GEAR; 4. Premere tre volte in successione DOWN A LEVEL; 5. Premere sul menù laterale - MORE OPTIONS; 6. Premere ROLL DICE; 7. Premere Okay.	Il Replay coincide con la fase di Capture
Test Case4_2	Funzionamento dei pulsanti per aggiungere e ridurre GEAR e LEVEL, lancio combinatorio.	WVGA (Nexus S) - Smartphone	Portrait Lingua italiana	API 26	1. Premere tre volte in successione UP A LEVEL; 2. Premere ADD GEAR; 3. Premere REMOVE GEAR; 4. Premere tre volte in successione DOWN A LEVEL; 5. Premere sul menù laterale - MORE OPTIONS; 6. Premere ROLL DICE; 7. Premere Okay.	Il Replay coincide con la fase di Capture
Test Case4_3	Funzionamento dei pulsanti per aggiungere e ridurre GEAR e LEVEL, lancio combinatorio.	WVGA (Nexus S) - Smartphone	Portrait Lingua spagnola	API 26	1. Premere tre volte in successione UP A LEVEL; 2. Premere ADD GEAR; 3. Premere REMOVE GEAR; 4. Premere tre volte in successione DOWN A LEVEL; 5. Premere sul menù laterale - MORE OPTIONS; 6. Premere ROLL DICE; 7. Premere Okay.	Il Replay coincide con la fase di Capture

Fig. 3.13 Test Case 4

Il quinto ed ultimo scenario di utilizzo che prevediamo consiste nella possibilità di aumentare e diminuire nuovamente il livello, dopo aver modificato le impostazioni dell'applicazione tramite l'ausilio del menù laterale. Essendo l'applicazione customizzata per dispositivi Android piuttosto obsoleti che prevedevano il back button fisico, la stessa applicazione lanciata su dispositivi Android più moderni non prevede più l'utilizzo dello stesso bottone (non essendo più presente) e richiederebbe un comando back per poter tornare alla pagina principale dell'app. In questo caso, rinunciamo parzialmente alla fase di Capture ed aggiungiamo una stringa di codice, che bisogna istanziare ogni volta, la quale ci consente di simulare il back button. In particolare, il test è stato registrato su un WVGA (Nexus S) con API 26, in modalità portrait. Gli step da eseguire sono i seguenti:

1. Premere tre volte in successione sul bottone UP A LEVEL;
2. Premere una volta su ADD GEAR;
3. Premere su REMOVE GEAR;
4. Premere su MORE OPTIONS;
5. Premere su RESET;
6. Premere su MORE OPTIONS;
7. Premere CHANGE SETTINGS;
8. Togliere la spunta su KEEP SCREEN AWAKE;
9. Rimettere la spunta su KEEP SCREEN AWAKE;
10. Togliere la spunta su VICTORY DIALOG;
11. Rimettere la spunta su VICTORY DIALOG;
12. Premere su MAX LEVEL;
13. Premere OK;
14. Premere su UP A LEVEL.

Di seguito, si riporta porzione del codice corrispondente al quinto test case:

```
1.  @Test
2.   public void sampleTest() throws InterruptedException{
3.     MobileElement el1 = (MobileElement) driver.findElementById("info.bpace.munchlife:id/up_
    button");
4.     el1.click();
5.     MobileElement el2 = (MobileElement) driver.findElementById("info.bpace.munchlife:id/up_
    button");
6.     el2.click();
7.     MobileElement el3 = (MobileElement) driver.findElementById("info.bpace.munchlife:id/up_
    button");
8.     el3.click();
9.     MobileElement el4 = (MobileElement) driver.findElementById("info.bpace.munchlife:id/up_
    gear_button");
10.    el4.click();
11.    MobileElement el5 = (MobileElement) driver.findElementById("info.bpace.munchlife:id/dow
    n_gear_button");
12.    el5.click();
13.    MobileElement el6 = (MobileElement) driver.findElementByAccessibilityId("More options")
    ;
14.    el6.click();
15.    Thread.sleep(2000);
16.    MobileElement el7 = (MobileElement) driver.findElementByXPath("/hierarchy/android.widge
    t.FrameLayout/android.widget.FrameLayout/android.widget.ListView/android.widget.LinearLayo
    ut[1]/android.widget.RelativeLayout/android.widget.TextView");
17.    el7.click();
18.    MobileElement el8 = (MobileElement) driver.findElementByAccessibilityId("More options")
    ;
19.    el8.click();
20.    Thread.sleep(2000);
21.    MobileElement el9 = (MobileElement) driver.findElementByXPath("/hierarchy/android.widge
    t.FrameLayout/android.widget.FrameLayout/android.widget.ListView/android.widget.LinearLayo
    ut[3]/android.widget.RelativeLayout/android.widget.TextView");
22.    el9.click();
23.    MobileElement el10 = (MobileElement) driver.findElementByXPath("/hierarchy/android.widg
    et.FrameLayout/android.view.ViewGroup/android.widget.FrameLayout[2]/android.widget.LinearL
    ayout/android.widget.LinearLayout/android.widget.LinearLayout/android.widget.ListView/andr
    oid.widget.LinearLayout[1]/android.widget.LinearLayout/android.widget.CheckBox");
24.    el10.click();
25.    el10.click();
26.    MobileElement el11 = (MobileElement) driver.findElementByXPath("/hierarchy/android.widg
    et.FrameLayout/android.view.ViewGroup/android.widget.FrameLayout[2]/android.widget.LinearL
    ayout/android.widget.LinearLayout/android.widget.LinearLayout/android.widget.ListView/andr
    oid.widget.LinearLayout[2]/android.widget.LinearLayout/android.widget.CheckBox");
27.    el11.click();
28.    el11.click();
29.    MobileElement el12 = (MobileElement) driver.findElementByXPath("/hierarchy/android.widg
    et.FrameLayout/android.view.ViewGroup/android.widget.FrameLayout[2]/android.widget.LinearL
    ayout/android.widget.LinearLayout/android.widget.LinearLayout/android.widget.ListView/andr
    oid.widget.LinearLayout[3]/android.widget.RelativeLayout/android.widget.TextView[2]");
30.    el12.click();
31.    MobileElement el13 = (MobileElement) driver.findElementById("android:id/button1");
32.    el13.click();
33.    MobileElement el14 = (MobileElement) driver.findElementByXPath("/hierarchy/android.widg
    et.FrameLayout/android.view.ViewGroup/android.widget.FrameLayout[1]/android.view.ViewGroup
    /android.widget.TextView");
34.    el14.click();
35.    driver.navigate().back();
36.    MobileElement el15 = (MobileElement) driver.findElementById("info.bpace.munchlife:id/up
    _gear_button");
37.    el15.click();
38.  }
```

Fig. 3.14 Codice corrispondente agli step previsti dal quinto Test Case.

La stringa *driver.navigate().back()*, inserita nel codice, ci consente di simulare il back button laddove non presente. Il test è stato, poi, rieseguito in Eclipse sui seguenti dispositivi con descrizione dell'esito del test:

- WVGA (Nexus S) con API 26, modalità portrait (stesso device): il replay viene eseguito correttamente;
- WVGA (Nexus S) con API 23, modalità portrait (stesso device): il replay viene eseguito correttamente;
- Galaxy Nexus con API 23, modalità portrait (device differente): il replay viene eseguito correttamente;
- Nexus 10 con API 24, modalità portrait, device differente (Tablet): il replay viene eseguito correttamente;

ID	Descrizione	ıratteristiche Emulato	Modalità	Versione di Android	Input	Esito
Test Case5_1	Funzionamento dei pulsanti per aggiungere e ridurre GEAR e LEVEL, RESET e CHANGE SETTINGS.	WVGA (Nexus S) - Smartphone	Portrait	API 26	1. Premere tre volte in successione UP A LEVEL; 2. Premere ADD GEAR; 3. Premere REMOVE GEAR; 4. Premere sul menù laterale - MORE OPTIONS; 5. Premere su RESET; 6. Premere sul menù laterale - MORE OPTIONS; 7. Premere CHANGE SETTINGS; 8. Togliere la spunta su KEEP SCREEN AWAKE; 9. Rimettere la spunta su KEEP SCREEN AWAKE. 10. Togliere la spunta su VICTORY DIALOG; 11. Rimettere la spunta su VICTORY DIALOG; 12. Premere MAX LEVEL; 13. Premere OK; 14. Premere UP A LEVEL.	Il Replay coincide con la fase di Capture
Test Case5_2	Funzionamento dei pulsanti per aggiungere e ridurre GEAR e LEVEL, RESET e CHANGE SETTINGS.	WVGA (Nexus S) - Smartphone	Portrait	API 23	1. Premere tre volte in successione UP A LEVEL; 2. Premere ADD GEAR; 3. Premere REMOVE GEAR; 4. Premere sul menù laterale - MORE OPTIONS; 5. Premere su RESET; 6. Premere sul menù laterale - MORE OPTIONS; 7. Premere CHANGE SETTINGS; 8. Togliere la spunta su KEEP SCREEN AWAKE; 9. Rimettere la spunta su KEEP SCREEN AWAKE. 10. Togliere la spunta su VICTORY DIALOG; 11. Rimettere la spunta su VICTORY DIALOG; 12. Premere MAX LEVEL; 13. Premere OK; 14. Premere UP A LEVEL.	Il Replay coincide con la fase di Capture
Test Case5_3	Funzionamento dei pulsanti per aggiungere e ridurre GEAR e LEVEL, RESET e CHANGE SETTINGS.	Galaxy Nexus - Smartphone	Portrait	API 23	1. Premere tre volte in successione UP A LEVEL; 2. Premere ADD GEAR; 3. Premere REMOVE GEAR; 4. Premere sul menù laterale - MORE OPTIONS; 5. Premere su RESET; 6. Premere sul menù laterale - MORE OPTIONS; 7. Premere CHANGE SETTINGS; 8. Togliere la spunta su KEEP SCREEN AWAKE; 9. Rimettere la spunta su KEEP SCREEN AWAKE. 10. Togliere la spunta su VICTORY DIALOG; 11. Rimettere la spunta su VICTORY DIALOG; 12. Premere MAX LEVEL; 13. Premere OK; 14. Premere UP A LEVEL.	Il Replay coincide con la fase di Capture
Test Case5_4	Funzionamento dei pulsanti per aggiungere e ridurre GEAR e LEVEL, RESET e CHANGE SETTINGS.	Nexus 10 - Tablet	Portrait	API 24	1. Premere tre volte in successione UP A LEVEL; 2. Premere ADD GEAR; 3. Premere REMOVE GEAR; 4. Premere sul menù laterale - MORE OPTIONS; 5. Premere su RESET; 6. Premere sul menù laterale - MORE OPTIONS; 7. Premere CHANGE SETTINGS; 8. Togliere la spunta su KEEP SCREEN AWAKE; 9. Rimettere la spunta su KEEP SCREEN AWAKE. 10. Togliere la spunta su VICTORY DIALOG; 11. Rimettere la spunta su VICTORY DIALOG; 12. Premere MAX LEVEL; 13. Premere OK; 14. Premere UP A LEVEL.	Il Replay coincide con la fase di Capture

Fig. 3.15 Test Case 5

3.3 Trolly

Un'altra app sulla quale è stata basata la nostra analisi è Trolly. Si tratta di un'app Android open source per la lista della spesa che ha come caratteristica quella di essere un'applicazione molto semplice senza funzioni fastidiose o complicate. L'ultima release risale al 2010. Anche in questo caso, si tratta di un'app piuttosto obsoleta.

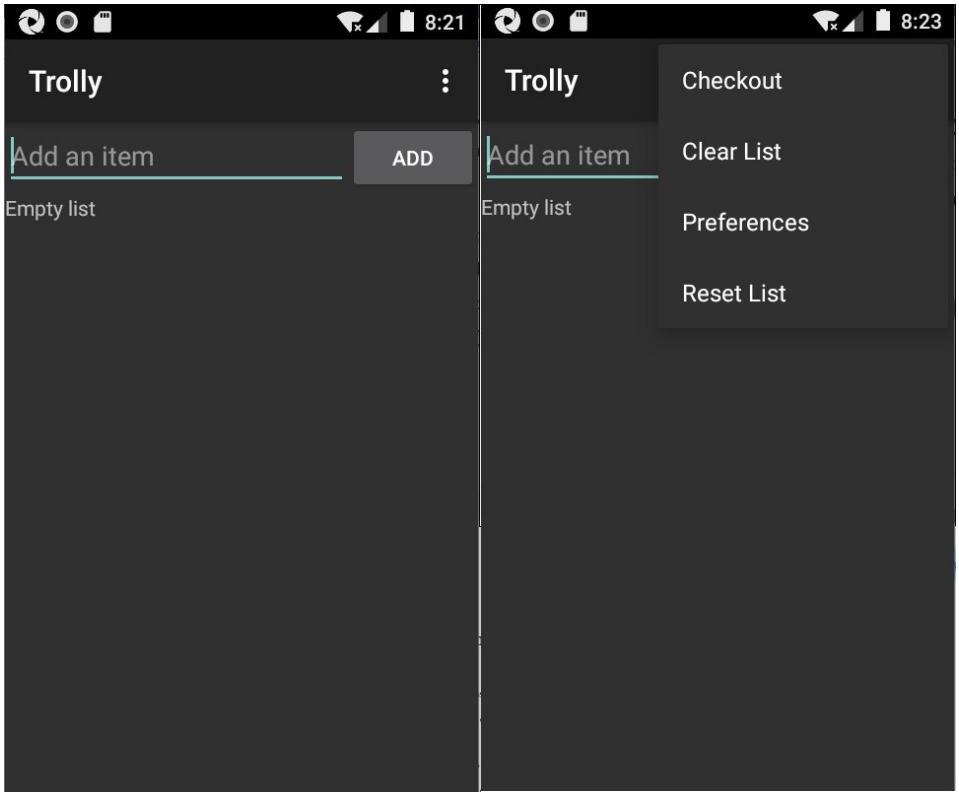


Fig. 3.16 Trolly

3.3.1 Test Case 1

Il primo scenario di test consiste nel verificare la possibilità di inserire tre item all'interno della lista della spesa. Il test case appena descritto prevede le seguenti azioni:

- Nel textbox, inserire la parola "Patate".
- Premere su "add".
- Nel textbox inserire la parola "Cipolle".
- Premere su "add".
- Nel textbox, inserire la parola "Carote".
- Premere su "add".

Di seguito, la porzione di codice corrispondente alle azioni precedenti:

```
1.  @Test
2.   public void sampleTest() {
3.     MobileElement el2 = (MobileElement) driver.findElementById("caldwell.ben.trolly:id/text
    box");
4.     el2.click();
5.     el2.sendKeys("Patate");
6.     MobileElement el3 = (MobileElement) driver.findElementById("caldwell.ben.trolly:id/btn_
    add");
7.     el3.click();
8.     MobileElement el4 = (MobileElement) driver.findElementById("caldwell.ben.trolly:id/text
    box");
9.     el4.sendKeys("Cipolle");
10.    MobileElement el5 = (MobileElement) driver.findElementById("caldwell.ben.trolly:id/btn_
    add");
11.    el5.click();
12.    el5.click();
13.    MobileElement el6 = (MobileElement) driver.findElementById("caldwell.ben.trolly:id/text
    box");
14.    el6.sendKeys("Carote");
15.    MobileElement el7 = (MobileElement) driver.findElementById("caldwell.ben.trolly:id/btn_
    add");
16.    el7.click();
17.  }
```

Fig. 3.17 Individuazione e click dei pulsanti corrispondenti alle azioni del Test Case 1

Il test è stato registrato in Appium su un WGA (Nexus S) con API 26, in modalità portrait, e successivamente rieseguito in Eclipse sullo stesso vendor. Anche qui, le azioni che coinvolgono il click dei pulsanti vengono interpretate correttamente. Di seguito, gli emulatori utilizzati con una breve descrizione dell'esito del test:

- WVGA (Nexus S) API 26, stesso emulatore, modalità portrait: il replay viene eseguito correttamente
- Nexus 10 API 23, modalità portrait, device diverso (Tablet): il replay avviene correttamente.
- Nexus 10 API 23, modalità landscape, device diverso (Tablet): il replay avviene correttamente.
- WVGA (Nexus S) API 26, stesso emulatore, modalità landscape: il replay viene eseguito correttamente.

ID	Descrizione	Caratteristiche Emulatore	Modalità	Versione di Android	Input	Esito
Test Case1_1	Inserimento di item all'interno della lista della spesa	WVGA (Nexus S) - Smartphone	Portrait	API 26	1. Nel textbox, inserire la parola PATATE; 2. Premere ADD; 3. Nel textbox, inserire la parola CIPOLLE; 4. Premere ADD; 5. Nel textbox, inserire la parola CAROTE; 6. Premere ADD;	Il Replay coincide con la fase di Capture
Test Case1_2	Inserimento di item all'interno della lista della spesa	WVGA (Nexus S) - Smartphone	Landscape	API 26	1. Nel textbox, inserire la parola PATATE; 2. Premere ADD; 3. Nel textbox, inserire la parola CIPOLLE; 4. Premere ADD; 5. Nel textbox, inserire la parola CAROTE; 6. Premere ADD;	Il Replay coincide con la fase di Capture
Test Case1_3	Inserimento di item all'interno della lista della spesa	Nexus 10 - Tablet	Portrait	API 23	1. Nel textbox, inserire la parola PATATE; 2. Premere ADD; 3. Nel textbox, inserire la parola CIPOLLE; 4. Premere ADD; 5. Nel textbox, inserire la parola CAROTE; 6. Premere ADD;	Il Replay coincide con la fase di Capture
Test Case1_4	Inserimento di item all'interno della lista della spesa	Nexus 10 - Tablet	Landscape	API 23	1. Nel textbox, inserire la parola PATATE; 2. Premere ADD; 3. Nel textbox, inserire la parola CIPOLLE; 4. Premere ADD; 5. Nel textbox, inserire la parola CAROTE; 6. Premere ADD;	Il Replay coincide con la fase di Capture

Fig. 3.18 Test Case 1

3.3.2 Test Case 2

Il secondo scenario di utilizzo che prevediamo consiste nel registrare un test su di un dispositivo in lingua italiana e, verificare la sua replicabilità, rieseguendo il medesimo test su un dispositivo in lingua inglese e spagnola. Dunque, in primis, è necessario settare la lingua di interesse, nel nostro caso l'italiano, dai system settings del device. In particolare, il test è stato registrato su un WVGA (Nexus S) con API 26, in modalità portrait. Gli step da eseguire sono i seguenti:

1. Nel textbox, inserire la parola "Sedano";
2. Premere su ADD;
3. Nel textbox, inserire la parola "Rapa";
4. Cancellare il secondo elemento della lista;
5. Premere sui tre puntini corrispondenti al menù laterale – ALTRE OPZIONI;
6. Premere su CHECK OUT;

Di seguito, si riporta porzione del codice corrispondente alle azioni appena descritte:

```
1.  @Test
2.    public void sampleTest()throws InterruptedException {
3.      MobileElement el1 = (MobileElement) driver.findElementById("caldwell.ben.trolly:id/tex
    tbox");
4.      el1.sendKeys("Sedano");
5.      MobileElement el2 = (MobileElement) driver.findElementById("caldwell.ben.trolly:id/btn
    _add");
6.      el2.click();
7.      MobileElement el3 = (MobileElement) driver.findElementById("caldwell.ben.trolly:id/tex
    tbox");
8.      el3.sendKeys("Rapa");
9.      el3.click();
```

```
10.     MobileElement el4 = (MobileElement) driver.findElementByXPath("/hierarchy/android.widg
    et.FrameLayout/android.view.ViewGroup/android.widget.FrameLayout[2]/android.widget.LinearL
    ayout/android.widget.ListView/android.widget.TextView[2]");
11.     el4.click();
12.     MobileElement el5 = (MobileElement) driver.findElementByAccessibilityId("Altre opzioni
    ");
13.     el5.click();
14.     Thread.sleep(2000);
15.     MobileElement el6 = (MobileElement) driver.findElementByXPath("/hierarchy/android.widg
    et.FrameLayout/android.widget.FrameLayout/android.widget.ListView/android.widget.LinearLay
    out[1]/android.widget.RelativeLayout/android.widget.TextView");
16.     el6.click();
17.   }
```

Fig. 3.19 Individuazione e click dei pulsanti corrispondenti alle azioni del Test Case 2.

Il test è stato, poi, rieseguito in Eclipse sui seguenti dispositivi con descrizione dell'esito del test:

- WVGA (Nexus S) con API 26, in lingua inglese, modalità portrait (stesso device): il replay non viene eseguito correttamente. In particolare, il test fallisce allo step 5 in quanto il sistema non riesce ad identificare l'oggetto ALTRE OPZIONI che, in inglese, viene indicato con un nome differente.
- WVGA (Nexus S) con API 26, in lingua italiana, modalità portrait (stesso device): il replay viene eseguito correttamente;
- WVGA (Nexus S) con API 26, in lingua spagnola, modalità portrait (stesso device): il replay non viene eseguito correttamente. Anche qui, come nel caso duale in lingua inglese, il test fallisce allo step 5, in quanto il sistema non riesce ad identificare l'oggetto ALTRE OPZIONI, che, in lingua spagnola, viene indicato con un nome differente.

A valle di questi test possiamo asserire che, nel caso di cambio lingua, i test falliscono in quanto i locatori che individuano gli elementi dipendono dagli stessi test scritti sull'interfaccia, che cambia con la lingua. In seguito, verrà approfondito l'argomento con maggior dettaglio.

ID	Descrizione	Caratteristiche Emulatore	Modalità	Versione di Android	Input	Esito
Test Case2_1	Inserimento di item all'interno della lista della spesa e CHECKOUT.	WVGA (Nexus S) - Smartphone	Portrait Lingua Inglese	API 26	1. Nel textbox, inserire la parola SEDANO; 2. Premere ADD; 3. Nel textbox, inserire la parola RAPA; 4. Premere ADD; 5. Premere sul menù laterale - MORE OPTIONS; 6. Premere CHECK OUT;	Non viene identificato l'oggetto MORE OPTIONS.
Test Case2_2	Inserimento di item all'interno della lista della spesa e CHECKOUT.	WVGA (Nexus S) - Smartphone	Portrait Lingua Italiana	API 26	1. Nel textbox, inserire la parola SEDANO; 2. Premere ADD; 3. Nel textbox, inserire la parola RAPA; 4. Premere ADD; 5. Premere sul menù laterale - MORE OPTIONS; 6. Premere CHECK OUT;	Il Replay coincide con la fase di Capture
Test Case2_3	Inserimento di item all'interno della lista della spesa e CHECKOUT.	WVGA (Nexus S) - Smartphone	Portrait Lingua Spagnola	API 28	1. Nel textbox, inserire la parola SEDANO; 2. Premere ADD; 3. Nel textbox, inserire la parola RAPA; 4. Premere ADD; 5. Premere sul menù laterale - MORE OPTIONS; 6. Premere CHECK OUT;	Non viene identificato l'oggetto MORE OPTIONS.

Fig. 3.20 Test Case 2

3.3.3 Test Case 3

Il terzo scenario di test consiste nell'inserire tre elementi nella lista della spesa, di cancellare il primo, il terzo ed il quarto; di abilitare nuovamente il primo. Inoltre, si vuole testare la possibilità di ripulire la lista; di aggiungere, a seguito, altri due elementi alla lista stessa ed, infine, di fare il checkout. Il test è stato registrato su un WVGA (Nexus S) con API 23, in modalità portrait. Gli step corrispondenti alle azioni pocanzi descritte sono i seguenti:

1. Nel textbox, inserire la parola "Pomodori".
2. Premere su ADD;
3. Nel textbox, inserire la parola "Peperoni";
4. Premere su ADD;
5. Nel textbox, inserire la parola "Rape";
6. Premere su ADD;
7. Barrare il primo elemento della lista;
8. Barrare il terzo elemento della lista;
9. Barrare il quarto elemento della lista;
10. Sbarrare il primo elemento della lista;
11. Premere sui tre puntini corrispondenti al menù laterale – More Options;
12. Premere su CLEAR LIST;
13. Premere su CANCEL;
14. Premere su CLEAR LIST;
15. Premere su OK;

16. Nel textbox, inserisci la parola “Limoni”;
17. Premere su ADD;
18. Nel textbox, inserisci la parola “Cipolle”;
19. Premere su ADD;
20. Premere sui tre puntini corrispondenti al menù laterale – More Options;
21. Premere su CHECKOUT.

Di seguito, si riporta porzione del codice corrispondente alle azioni descritte:

```
@Test
  public void sampleTest() throws InterruptedException{
     MobileElement el1 = (MobileElement) driver.findElementById("caldwell.ben.trolly:id/textbox");
     el1.sendKeys("Pomodori");
     MobileElement el2 = (MobileElement) driver.findElementById("caldwell.ben.trolly:id/btn_add");
     el2.click();
     MobileElement el3 = (MobileElement) driver.findElementById("caldwell.ben.trolly:id/textbox");
     el3.sendKeys("Peperoni");
     MobileElement el4 = (MobileElement) driver.findElementById("caldwell.ben.trolly:id/btn_add");
     el4.click();
     MobileElement el5 = (MobileElement) driver.findElementById("caldwell.ben.trolly:id/textbox");
     el5.sendKeys("Rape");
     MobileElement el6 = (MobileElement) driver.findElementById("caldwell.ben.trolly:id/btn_add");
     el6.click();
     MobileElement el7 = (MobileElement) driver.findElementByXPath("/hierarchy/android.widget.FrameLayout/android.view.ViewGroup/android.widget.FrameLayout[2]/android.widget.LinearLayout/android.widget.ListView/android.widget.TextView[1]");
     el7.click();
     Thread.sleep(2000);
     MobileElement el8 = (MobileElement) driver.findElementByXPath("/hierarchy/android.widget.FrameLayout/android.view.ViewGroup/android.widget.FrameLayout[2]/android.widget.LinearLayout/android.widget.ListView/android.widget.TextView[4]");
     el8.click();
     MobileElement el9 = (MobileElement) driver.findElementByXPath("/hierarchy/android.widget.FrameLayout/android.view.ViewGroup/android.widget.FrameLayout[2]/android.widget.LinearLayout/android.widget.ListView/android.widget.TextView[3]");
     el9.click();
     MobileElement el10 = (MobileElement) driver.findElementByXPath("/hierarchy/android.widget.FrameLayout/android.view.ViewGroup/android.widget.FrameLayout[2]/android.widget.LinearLayout/android.widget.ListView/android.widget.TextView[1]");
     el10.click();
     MobileElement el11 = (MobileElement) driver.findElementByAccessibilityId("More options");
     el11.click();
     Thread.sleep(2000);
     MobileElement el12 = (MobileElement) driver.findElementByXPath("/hierarchy/android.widget.FrameLayout/android.widget.FrameLayout/android.widget.ListView/android.widget.LinearLayout[2]/android.widget.RelativeLayout/android.widget.TextView");
     el12.click();
     MobileElement el13 = (MobileElement) driver.findElementById("android:id/button2");
     el13.click();
     MobileElement el14 = (MobileElement) driver.findElementByAccessibilityId("More options");
     el14.click();
     Thread.sleep(2000);
```

```
34.     MobileElement el15 = (MobileElement) driver.findElementByXPath("/hierarchy/android.wid
    get.FrameLayout/android.widget.FrameLayout/android.widget.ListView/android.widget.LinearLa
    yout[2]/android.widget.RelativeLayout/android.widget.TextView");
35.     el15.click();
36.     MobileElement el16 = (MobileElement) driver.findElementById("android:id/button1");
37.     el16.click();
38.     Thread.sleep(2000);
39.     MobileElement el17 = (MobileElement) driver.findElementById("caldwell.ben.trolly:id/te
    xtbox");
40.     el17.sendKeys("Limoni");
41.     MobileElement el18 = (MobileElement) driver.findElementById("caldwell.ben.trolly:id/bt
    n_add");
42.     el18.click();
43.     MobileElement el19 = (MobileElement) driver.findElementById("caldwell.ben.trolly:id/te
    xtbox");
44.     el19.sendKeys("Cipolle");
45.     MobileElement el20 = (MobileElement) driver.findElementById("caldwell.ben.trolly:id/it
    em");
46.     el20.click();
47.     MobileElement el21 = (MobileElement) driver.findElementById("caldwell.ben.trolly:id/bt
    n_add");
48.     el21.click();
49.     MobileElement el22 = (MobileElement) driver.findElementByAccessibilityId("More options
    ");
50.     el22.click();
51.     Thread.sleep(2000);
52.     MobileElement el23 = (MobileElement) driver.findElementByXPath("/hierarchy/android.wid
    get.FrameLayout/android.widget.FrameLayout/android.widget.ListView/android.widget.LinearLa
    yout[1]/android.widget.RelativeLayout/android.widget.TextView");
53.     el23.click();
54.   }
```

Fig. 3.21 Individuazione e click dei pulsanti corrispondenti alle azioni del Test Case 3.

Il test è stato, poi, rieseguito in Eclipse sui seguenti dispositivi con descrizione dell'esito del test:

- WVGA (Nexus S), stesso device, modalità portrait: il replay viene eseguito correttamente;
- Tablet Nexus 10 con API 24, modalità portrait, device differente (Tablet): il replay viene eseguito correttamente;
- WVGA (Nexus S) con API 26, modalità portrait, device differente (vendor diverso): il replay non avviene correttamente. Il test fallisce al punto 8 in quanto, dall'inizio del test, compare la keyboard del cellulare, che non scompare se non dismessa manualmente, la quale va a coprire la lista degli item. In particolare, il test fallisce al punto 8 dal momento che, vista la presenza della keyboard che copre la lista degli item, il sistema non riesce ad identificare il quarto elemento della lista.

ID	Descrizione	Caratteristiche Emulatore	Modalità	Versione di Android	Input	Esito
Test Case3_1	Inserimento di item all'interno della lista della spesa e CHECKOUT.	WVGA (Nexus S) - Smartphone	Portrait	API 23	1. Nel textbox, inserire la parola SEDANO; 2. Premere ADD; 3. Nel textbox, inserire la parola RAPA; 4. Premere ADD; 5. Premere sul menù laterale - MORE OPTIONS; 6. Premere CHECK OUT;	Il Replay coincide con la fase di Capture
Test Case3_2	Inserimento di item all'interno della lista della spesa e CHECKOUT.	Nexus 10 - Tablet	Portrait	API 24	1. Nel textbox, inserire la parola SEDANO; 2. Premere ADD; 3. Nel textbox, inserire la parola RAPA; 4. Premere ADD; 5. Premere sul menù laterale - MORE OPTIONS; 6. Premere CHECK OUT;	Il Replay coincide con la fase di Capture
Test Case3_3	Inserimento di item all'interno della lista della spesa e CHECKOUT.	WVGA (Nexus S) - Smartphone	Portrait Lingua Inglese	API 26	1. Nel textbox, inserire la parola SEDANO; 2. Premere ADD; 3. Nel textbox, inserire la parola RAPA; 4. Premere ADD; 5. Premere sul menù laterale - MORE OPTIONS; 6. Premere CHECK OUT;	Non compare la keyboard.

Fig. 3.22 Test Case 3

3.3.4 Test Case 4

Il quarto ed ultimo scenario di utilizzo che prevediamo consiste nella possibilità di inserire altri item all'interno della lista della spesa, dopo aver modificato le impostazioni dell'applicazione tramite l'ausilio del menù laterale. Anche qui, come per MunchLife, essendo l'applicazione customizzata per dispositivi Android piuttosto obsoleti che prevedevano il back button fisico, la stessa applicazione lanciata su dispositivi Android più moderni non prevede più l'utilizzo dello stesso bottone (non essendo più presente) e richiederebbe un comando back per poter tornare alla pagina principale dell'app. In questo caso, rinunciamo parzialmente alla fase di capture ed aggiungiamo una stringa di codice, che bisogna istanziare ogni volta, la quale ci consente di simulare il back button. In particolare, il test è stato registrato su un WVGA (Nexus S) con API 26, in modalità portrait. Gli step da eseguire sono i seguenti:

1. Nel textbox, inserire la parola "Pomodori";
2. Premere su ADD;
3. Nel textbox, inserire la parola "Minestrone";
4. Premere su ADD;
5. Cliccare sui tre puntini corrispondenti al menù laterale – MORE OPTIONS;
6. Cliccare su PREFERENCES;
7. Premere su LIST MODE SORT ORDER;
8. Premere su ALPHABETICAL DESCENDING;
9. Premere su SHOPPING MODE SORT ORDER;
10. Premere su ALPHABETICAL ASCENDING;
11. Nel textbox, inserire la parola "Patate";
12. Premere su ADD.

Di seguito, si riporta porzione del codice corrispondente alle azioni descritte:

```
1.  @Test
2.   public void sampleTest()throws InterruptedException {
3.     MobileElement el1 = (MobileElement) driver.findElementById("caldwell.ben.trolly:id/text
    box");
4.     el1.sendKeys("Pomodori");
5.     MobileElement el2 = (MobileElement) driver.findElementById("caldwell.ben.trolly:id/btn_
    add");
6.     el2.click();
7.     MobileElement el3 = (MobileElement) driver.findElementById("caldwell.ben.trolly:id/text
    box");
8.     el3.sendKeys("Minestrone");
9.     MobileElement el4 = (MobileElement) driver.findElementById("caldwell.ben.trolly:id/btn_
    add");
10.    el4.click();
11.    MobileElement el5 = (MobileElement) driver.findElementByAccessibilityId("More options")
    ;
12.    el5.click();
13.    Thread.sleep(2000);
14.    MobileElement el6 = (MobileElement) driver.findElementByXPath("/hierarchy/android.widge
    t.FrameLayout/android.widget.FrameLayout/android.widget.ListView/android.widget.LinearLayo
    ut[3]/android.widget.RelativeLayout/android.widget.TextView");
15.    el6.click();
16.    MobileElement el7 = (MobileElement) driver.findElementByXPath("/hierarchy/android.widge
    t.FrameLayout/android.view.ViewGroup/android.widget.FrameLayout[2]/android.widget.LinearLa
    yout/android.widget.LinearLayout/android.widget.LinearLayout/android.widget.ListView/andro
    id.widget.LinearLayout[1]/android.widget.RelativeLayout/android.widget.TextView[2]");
17.    el7.click();
18.    MobileElement el8 = (MobileElement) driver.findElementByXPath("/hierarchy/android.widge
    t.FrameLayout/android.widget.FrameLayout/android.widget.FrameLayout/android.widget.LinearL
    ayout/android.widget.FrameLayout/android.widget.ListView/android.widget.CheckedTextView[2]
    ");
19.    el8.click();
20.    MobileElement el9 = (MobileElement) driver.findElementByXPath("/hierarchy/android.widge
    t.FrameLayout/android.view.ViewGroup/android.widget.FrameLayout[2]/android.widget.LinearLa
    yout/android.widget.LinearLayout/android.widget.LinearLayout/android.widget.ListView/andro
    id.widget.LinearLayout[2]/android.widget.RelativeLayout/android.widget.TextView[2]");
21.    el9.click();
22.    MobileElement el10 = (MobileElement) driver.findElementByXPath("/hierarchy/android.widg
    et.FrameLayout/android.widget.FrameLayout/android.widget.FrameLayout/android.widget.Linear
    Layout/android.widget.FrameLayout/android.widget.ListView/android.widget.CheckedTextView[1
    ]");
23.    el10.click();
24.    driver.navigate().back();
25.    MobileElement el11 = (MobileElement) driver.findElementById("caldwell.ben.trolly:id/tex
    tbox");
26.    el11.sendKeys("Patate");
27.    MobileElement el12 = (MobileElement) driver.findElementById("caldwell.ben.trolly:id/btn
    _add");
28.    el12.click();
29.  }
```

Fig. 3.23 Individuazione e click dei pulsanti corrispondenti alle azioni del Test Case 4.

Il test è stato, poi, rieseguito in Eclipse sui seguenti dispositivi con descrizione dell'esito del test:

- WVGA (Nexus S) con API 26, stesso device, modalità portrait: il replay viene eseguito correttamente;
- Tablet Nexus 10 con API 24, modalità portrait, device differente (Tablet): il replay viene eseguito correttamente;

- WVGA (Nexus S) con API 23, modalità portrait, stesso device: il replay viene eseguito correttamente.

ID	Descrizione	Caratteristiche Emulatore	Modalità	Versione di Android	Input	Esito
Test Case4_1	Inserimento item nella lista della spesa e Preferences.	WVGA (Nexus S) - Smartphone	Portrait	API 26	1.Nel textbox, scrivere POMODORI; 2. Premere su ADD; 3. Nel textbox, scrivere MINESTRONE; 4. Premere su ADD; 5. Premere sul menù laterale - MORE OPTIONS; 6. Cliccare su PREFERENCES; 7. Premere su LIST MODE SORT ORDER; 8. Premere su ALPHABETICAL DESCENDING; 9. Premere su SHOPPING MODE SORT ORDER. 10. Premere su ALPHABETICAL ASCENDING; 11. Nel textbox, scrivere PATATE; 12. Premere ADD;	Il Replay coincide con la fase di Capture.
Test Case4_2	Inserimento item nella lista della spesa e Preferences.	WVGA (Nexus S) - Smartphone	Portrait	API 23	1.Nel textbox, scrivere POMODORI; 2. Premere su ADD; 3. Nel textbox, scrivere MINESTRONE; 4. Premere su ADD; 5. Premere sul menù laterale - MORE OPTIONS; 6. Cliccare su PREFERENCES; 7. Premere su LIST MODE SORT ORDER; 8. Premere su ALPHABETICAL DESCENDING; 9. Premere su SHOPPING MODE SORT ORDER. 10. Premere su ALPHABETICAL ASCENDING; 11. Nel textbox, scrivere PATATE; 12. Premere ADD;	Il Replay coincide con la fase di Capture.
Test Case4_3	Inserimento item nella lista della spesa e Preferences.	Nexus 10 - Tablet	Portrait	API 24	1.Nel textbox, scrivere POMODORI; 2. Premere su ADD; 3. Nel textbox, scrivere MINESTRONE; 4. Premere su ADD; 5. Premere sul menù laterale - MORE OPTIONS; 6. Cliccare su PREFERENCES; 7. Premere su LIST MODE SORT ORDER; 8. Premere su ALPHABETICAL DESCENDING; 9. Premere su SHOPPING MODE SORT ORDER. 10. Premere su ALPHABETICAL ASCENDING; 11. Nel textbox, scrivere PATATE; 12. Premere ADD;	Il Replay coincide con la fase di Capture.

Fig. 3.24 Test Case 4

3.4 DIAB

DIAB è un'applicazione open source, pensata per pazienti diabetici, che aiuta a gestire il diabete tenendo traccia dei valori di glucosio e delle iniezioni di insulina [17]. Utilizzando i dati generati all'interno dell'app, è possibile realizzare un plug-in personalizzato, il quale, una volta applicato all'app, fornirà informazioni intelligenti per i dosaggi di insulina, in base al contesto, in tempo reale. È possibile inoltre esportare i dati come un foglio di lavoro. Infine, è possibile integrare l'app con altri servizi di fitness per condividere dati, come Google Fit. L'ultima release dell'applicazione risale ad aprile 2020, quindi, si tratta di un'app più recente rispetto a quelle di Trolly e Munchlife, pocanzi descritte.

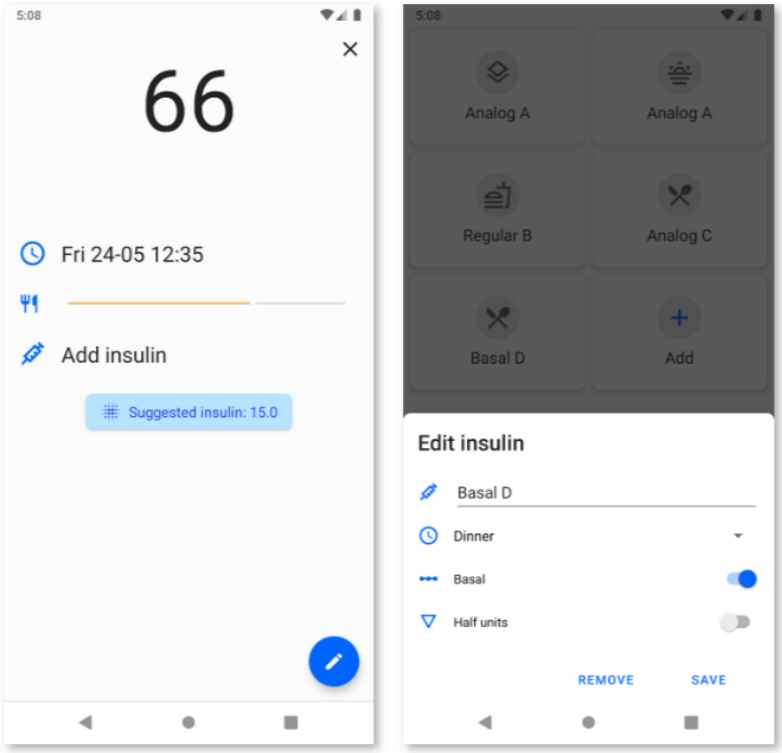


Fig. 3.25 Diab

3.4.1 Test Case 1

Il primo scenario di test che prevediamo consiste nell'inserire un nuovo valore di glucosio ad una determinata ora ed in un determinato giorno. Gli step che in questo scenario prevediamo sono i seguenti:

1. Premere sul bottone blu "Aggiungi nuovo valore";
2. Cliccare sull' editor time;
3. Cliccare su YESTERDAY;
4. Cliccare su 10;
5. Cliccare su 20;
6. Cliccare su OK;
7. Cliccare sulla spunta verde editor_fab;
8. Cliccare sulla X;

Il test è registrato su un WVGA (Nexus S) con API 23, in modalità portrait. Di seguito, viene riportato il codice generato dalle azioni appena descritte:

```
1.  package appium_diab;
2.
3.  import io.appium.java_client.MobileElement;
4.  import io.appium.java_client.android.AndroidDriver;
5.  import junit.framework.TestCase;
6.  import org.junit.After;
7.  import org.junit.Before;
8.  import org.junit.Test;
9.  import java.net.MalformedURLException;
10. import java.net.URL;
11. import org.openqa.selenium.remote.DesiredCapabilities;
12.
13. public class Sample_Test_Android6_Portrait {
14.
15.   private AndroidDriver driver;
16.
17.   @Before
18.   public void setUp() throws MalformedURLException {
19.     DesiredCapabilities desiredCapabilities = new DesiredCapabilities();
20.     desiredCapabilities.setCapability("platformName", "android");
21.     desiredCapabilities.setCapability("platformVersion", "6.0");
22.     desiredCapabilities.setCapability("deviceName", "Emulator");
23.     desiredCapabilities.setCapability("app", "C:\\Users\\Sistemitaly\\Desktop\\Tesi\\diab\
    \app\\build\\outputs\\apk\\googleFit\\debug\\app-googleFit-debug.apk");
24.     desiredCapabilities.setCapability("allowTestPackages", true);
25.     desiredCapabilities.setCapability("autoGrantPermissions", true);
26.     desiredCapabilities.setCapability("noReset", true);
27.
28.     URL remoteUrl = new URL("http://localhost:4723/wd/hub");
29.
30.     driver = new AndroidDriver(remoteUrl, desiredCapabilities);
31.   }
32.
33.   @Test
34.   public void sampleTest() throws InterruptedException {
35.     MobileElement el1 = (MobileElement) driver.findElementById("it.diab:id/item_glucose_in
    fo");
36.     el1.click();
37.     Thread.sleep(600);
38.     MobileElement el2 = (MobileElement) driver.findElementById("it.diab:id/editor_time");

39.     el2.click();
40.     MobileElement el3 = (MobileElement) driver.findElementByXPath("/hierarchy/android.widg
    et.FrameLayout/android.widget.LinearLayout/android.widget.FrameLayout/android.widget.Frame
    Layout/android.widget.FrameLayout/androidx.appcompat.widget.LinearLayoutCompat/android.wid
    get.FrameLayout/android.widget.ListView/android.widget.TextView[2]");
41.     el3.click();
42.     MobileElement el4 = (MobileElement) driver.findElementByAccessibilityId("10");
43.     el4.click();
44.     MobileElement el5 = (MobileElement) driver.findElementByAccessibilityId("20");
45.     el5.click();
46.     MobileElement el6 = (MobileElement) driver.findElementById("android:id/button1");
47.     Thread.sleep(600);
48.     el6.click();
49.     MobileElement el7 = (MobileElement) driver.findElementByXPath("/hierarchy/android.widg
    et.FrameLayout/android.widget.LinearLayout/android.widget.FrameLayout/android.widget.Linea
    rLayout/android.widget.FrameLayout/android.view.ViewGroup/android.widget.LinearLayout[1]/a
    ndroid.widget.LinearLayout/android.widget.TextView");
50.     el7.click();
51.     Thread.sleep(600);
52.     MobileElement el8 = (MobileElement) driver.findElementById("it.diab:id/editor_fab");
53.     el8.click();
```

```
54.   }
55.
56.   @After
57.   public void tearDown() {
58.     driver.quit();
59.   }
60. }
```

Fig. 3.26 Codice corrispondente alle azioni del Test Case 1.

Il test, poi, viene rieseguito in Eclipse sui seguenti dispositivi:

- WVGA (Nexus S) API 23, stesso emulatore, modalità portrait: il replay viene eseguito correttamente
- WVGA (Nexus S) API 23, stesso emulatore, modalità landscape; il test fallisce allo step 2 in quanto, non riesce a trovare il bottone editor_time che in modalità landscape si sovrappone alla tastiera, come si può facilmente notare nella seguente figura:

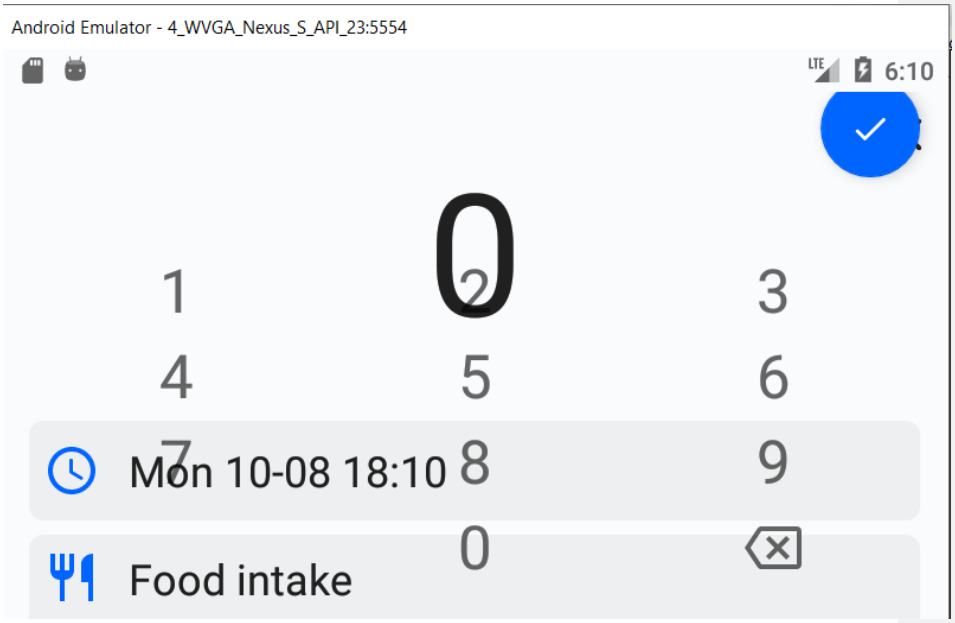


Fig. 3.27 DIAB in modalità landscape.

- WVGA (Nexus S) API 26, stesso emulatore, in modalità portrait: il replay viene eseguito correttamente.

- WVGA (Nexus S) API 26, stesso emulatore, in modalità landscape: anche qui, come nel duale emulatore con API 23, in modalità landscape, il test fallisce in quanto il sistema non è in grado di rilevare l'oggetto editor_time.

ID	Descrizione	Caratteristiche Emulatore	Modalità	Versione di Android	Input	Esito
Test Case3_1	Inserimento nuovo valore di glucosio in un determinato giorno ed a una determinata ora.	WVGA (Nexus S) - Smartphone	Portrait	API 23	1. Premere sul bottone blu AGGIUNGI NUOVO VALORE; 2. Premere sull'editor time; 3. Cliccare su YESTERDAY; 4. Cliccare su 10; 5. Cliccare su 20; 6. Cliccare su OK; 7. Cliccare sulla spunta verde editor_fab; 8. Cliccare sulla X.	Il Replay coincide con la fase di Capture.
Test Case3_2	Inserimento nuovo valore di glucosio in un determinato giorno ed a una determinata ora.	WVGA (Nexus S) - Smartphone	Landscape	API 23	1. Premere sul bottone blu AGGIUNGI NUOVO VALORE; 2. Premere sull'editor time; 3. Cliccare su YESTERDAY; 4. Cliccare su 10; 5. Cliccare su 20; 6. Cliccare su OK; 7. Cliccare sulla spunta verde editor_fab; 8. Cliccare sulla X.	Non viene identificato il bottone EDITOR TIME
Test Case3_3	Inserimento nuovo valore di glucosio in un determinato giorno ed a una determinata ora.	WVGA (Nexus S) - Smartphone	Portrait	API 26	1. Premere sul bottone blu AGGIUNGI NUOVO VALORE; 2. Premere sull'editor time; 3. Cliccare su YESTERDAY; 4. Cliccare su 10; 5. Cliccare su 20; 6. Cliccare su OK; 7. Cliccare sulla spunta verde editor_fab; 8. Cliccare sulla X.	Il Replay coincide con la fase di Capture.
Test Case3_4	Inserimento nuovo valore di glucosio in un determinato giorno ed a una determinata ora.	WVGA (Nexus S) - Smartphone	Landscape	API 26	1. Premere sul bottone blu AGGIUNGI NUOVO VALORE; 2. Premere sull'editor time; 3. Cliccare su YESTERDAY; 4. Cliccare su 10; 5. Cliccare su 20; 6. Cliccare su OK; 7. Cliccare sulla spunta verde editor_fab; 8. Cliccare sulla X.	Non viene identificato il bottone EDITOR TIME

Fig. 3.28 Test Case 3

3.4.2 Test Case 2

Il secondo scenario di utilizzo che prevediamo consiste nel registrare un test su di un dispositivo in lingua italiana e, verificare la sua replicabilità, rieseguendo il medesimo test su un dispositivo in lingua inglese e spagnola. Dunque, in primis, è necessario settare la lingua di interesse, nel nostro caso l'italiano, dai system settings del device. In particolare, il test è stato registrato su un WVGA (Nexus S) con API 23, in modalità portrait. Gli step da eseguire sono i seguenti:

1. Premere sul cerchietto arancione (item_glucose_status);
2. Premere su MODIFICA INSULINE;
3. Premere sul simbolo dell'ingranaggio;
4. Premere sul simbolo "+";
5. Nell' insulin_edit_name, inserire "Iniezione5";
6. Cliccare su "Extra";
7. Cliccare su "cena";
8. Switchare "basale" su on;
9. Switchare "mezze unità" su on;
10. Switchare "mezze unità" su off;

11. Cliccare su “Salva”;
12. Premere su “back” button;
13. Premere su “Modifica insuline”;
14. Aggiungi il valore 12;
15. Premere su “Applica”;
16. Premere su “X”;
17. Premere sul bottone blu “Aggiungi nuovo valore”;
18. Cliccare 1;
19. Cliccare 1;
20. Cliccare 0;
21. Cliccare sull’ editor time;
22. Cliccare su ALTRA DATA;
23. Cliccare su 20 agosto 2020;
24. Cliccare su OK;
25. Cliccare su 20;
26. Cliccare su 32;
27. Cliccare su OK;
28. Cliccare sulla spunta verde editor_fab;
29. Cliccare sulla X;

Di seguito, viene riportato il codice corrispondente alle azioni pocanzi descritte:

```
1.  package appium_diab_language;
2.
3.  import io.appium.java_client.MobileElement;
4.  import io.appium.java_client.android.AndroidDriver;
5.  import junit.framework.TestCase;
6.  import org.junit.After;
7.  import org.junit.Before;
8.  import org.junit.Test;
9.  import java.net.MalformedURLException;
10. import java.net.URL;
11. import org.openqa.selenium.remote.DesiredCapabilities;
12.
13. public class Sample_Test_Android6_Portrait {
14.
15.   private AndroidDriver driver;
16.
17.   @Before
18.   public void setUp() throws MalformedURLException {
19.     DesiredCapabilities desiredCapabilities = new DesiredCapabilities();
20.     desiredCapabilities.setCapability("platformName", "android");
21.     desiredCapabilities.setCapability("platformVersion", "6.0");
22.     desiredCapabilities.setCapability("deviceName", "Emulator");
23.     desiredCapabilities.setCapability("app", "C:\\Users\\Sistemitaly\\Desktop\\Tesi\\diab\
    \app\\build\\outputs\\apk\\googleFit\\debug\\app-googleFit-debug.apk");
```

```
24.     desiredCapabilities.setCapability("allowTestPackages", true);
25.     desiredCapabilities.setCapability("autoGrantPermissions", true);
26.     desiredCapabilities.setCapability("noReset", true);
27.
28.     URL remoteUrl = new URL("http://localhost:4723/wd/hub");
29.
30.     driver = new AndroidDriver(remoteUrl, desiredCapabilities);
31.   }
32.
33.   @Test
34.   public void sampleTest()throws InterruptedException {
35.     MobileElement el1 = (MobileElement) driver.findElementById("it.diab:id/item_glucose_st
    atus");
36.     el1.click();
37.     Thread.sleep(2000);
38.     MobileElement el2 = (MobileElement) driver.findElementById("it.diab:id/editor_insulin"
    );
39.     el2.click();
40.     MobileElement el3 = (MobileElement) driver.findElementByAccessibilityId("Modifica insu
    line");
41.     el3.click();
42.     Thread.sleep(2000);
43.     MobileElement el4 = (MobileElement) driver.findElementById("it.diab:id/insulin_fab");

44.     el4.click();
45.     Thread.sleep(2000);
46.     MobileElement el5 = (MobileElement) driver.findElementById("it.diab:id/insulin_edit_na
    me");
47.     el5.sendKeys("Iniezione5");
48.     MobileElement el6 = (MobileElement) driver.findElementById("it.diab:id/insulin_edit_ti
    me");
49.     el6.click();
50.     Thread.sleep(2000);
51.     MobileElement el7 = (MobileElement) driver.findElementByXPath("/hierarchy/android.widg
    et.FrameLayout/android.widget.FrameLayout/android.widget.ListView/android.widget.TextView[
    6]");
52.     el7.click();
53.     MobileElement el8 = (MobileElement) driver.findElementById("it.diab:id/insulin_edit_ba
    sal");
54.     el8.click();
55.     MobileElement el9 = (MobileElement) driver.findElementById("it.diab:id/insulin_edit_ha
    lf_units");
56.     el9.click();
57.     el9.click();
58.     MobileElement el10 = (MobileElement) driver.findElementById("it.diab:id/insulin_edit_s
    ave");
59.     el10.click();
60.     Thread.sleep(2000);
61.     MobileElement el11 = (MobileElement) driver.findElementByXPath("/hierarchy/android.wid
    get.FrameLayout/android.widget.LinearLayout/android.widget.FrameLayout/android.widget.Line
    arLayout/android.widget.FrameLayout/android.view.ViewGroup/android.view.ViewGroup/android.
    widget.ImageButton");
62.     el11.click();
63.     Thread.sleep(2000);
64.     MobileElement el12 = (MobileElement) driver.findElementById("it.diab:id/glucose_editor
    _insulin_value");
65.     el12.sendKeys("12");
66.     MobileElement el13 = (MobileElement) driver.findElementById("it.diab:id/glucose_editor
    _insulin_positive");
67.     el13.click();
68.     Thread.sleep(2000);
69.     MobileElement el14 = (MobileElement) driver.findElementByAccessibilityId("Chiudi");
70.     el14.click();
71.     Thread.sleep(2000);
72.     MobileElement el15 = (MobileElement) driver.findElementById("it.diab:id/fab");
73.     el15.click();
74.     Thread.sleep(2000);
```

```
75.     MobileElement el16 = (MobileElement) driver.findElementById("it.diab:id/keyboard_key_1
    ");
76.     el16.click();
77.     el16.click();
78.     MobileElement el17 = (MobileElement) driver.findElementById("it.diab:id/keyboard_key_0
    ");
79.     el17.click();
80.     MobileElement el18 = (MobileElement) driver.findElementById("it.diab:id/editor_time");

81.     el18.click();
82.     Thread.sleep(2000);
83.     MobileElement el19 = (MobileElement) driver.findElementByXPath("/hierarchy/android.wid
    get.FrameLayout/android.widget.LinearLayout/android.widget.FrameLayout/android.widget.Fram
    eLayout/android.widget.FrameLayout/androidx.appcompat.widget.LinearLayoutCompat/android.wi
    dget.FrameLayout/android.widget.ListView/android.widget.TextView[3]");
84.     el19.click();
85.     MobileElement el20 = (MobileElement) driver.findElementByAccessibilityId("20 agosto 20
    20");
86.     el20.click();
87.     MobileElement el21 = (MobileElement) driver.findElementById("android:id/button1");
88.     el21.click();
89.     Thread.sleep(2000);
90.     MobileElement el22 = (MobileElement) driver.findElementByAccessibilityId("20");
91.     el22.click();
92.     MobileElement el23 = (MobileElement) driver.findElementByAccessibilityId("35");
93.     el23.click();
94.     Thread.sleep(2000);
95.     MobileElement el24 = (MobileElement) driver.findElementById("android:id/button1");
96.     el24.click();
97.     Thread.sleep(2000);
98.     MobileElement el25 = (MobileElement) driver.findElementById("it.diab:id/editor_fab");

99.     el25.click();
100.          MobileElement el26 = (MobileElement) driver.findElementByAccessibilityId("Chiud
    i");
101.          el26.click();
102.        }
103.        @After
104.        public void tearDown() {
105.          driver.quit();
106.        }
107.      }
```

Fig. 3.29 Codice corrispondente alle azioni del Test Case 2.

Il test è stato poi rieseguito sui seguenti dispositivi:

- WVGA (Nexus S) API 23, modalità portrait, in lingua italiana: il replay viene eseguito correttamente;
- WVGA (Nexus S) API 23, medesimo dispositivo, modalità portrait, in lingua inglese: il test fallisce al punto 2 in quanto il bottone MODIFICA INSULINE non viene identificato. In lingua inglese, il medesimo oggetto viene identificato con l'id glucose_editor_insulin_value. Dunque, l'oggetto viene identificato con un id che ha un nome differente a seconda della lingua scelta nei system settings del device.
- WVGA (Nexus S) API 23, medesimo dispositivo, modalità portrait, in lingua spagnolo: il test fallisce al punto 2 per il medesimo motivo del precedente, ovvero il bottone MODIFICA

INSULINE non viene identificato, in quanto il test è registrato in una lingua diversa rispetto a quella con la quale viene rieseguito.

A valle di questi test possiamo asserire che nel caso di cambio lingua i test falliscono in quanto i locatori che individuano gli elementi dipendono dagli stessi test scritti sull'interfaccia, che cambia con la lingua. Tale inconveniente, teoricamente, potrebbe essere risolto. La soluzione consiste nell'indicare non direttamente il testo, ma il puntatore nel file di risorse, in modo tale che venga dinamicamente letto il valore della lingua, in quel momento, in funzione. Tuttavia, tale soluzione, sebbene valida ed applicabile nell'attuale versione di Espresso Test Recorder, non è comunque applicabile se si esegue la fase di Capture in assenza di codice sorgente, in quanto, a tempo di esecuzione la risorsa è visibile come valore e non come puntatore al file delle risorse. Nelle ipotesi in cui ci siamo posti, ovvero, in assenza di source code, ma in presenza del solo APK, come già detto, questa soluzione non è contemplata. Un'altra soluzione sarebbe quella di non utilizzare i testi nella query per localizzare gli elementi (ad esempio, in Espresso Test Recorder, c'è un'opzione per disabilitare l'utilizzo dei tasti), ma ovviamente le query che non li considerano sono molto meno potenti e hanno una maggior probabilità di fallire per numerose altre cause. Dunque, anche questa, non è una strada perseguibile.

ID	Descrizione	Caratteristiche Emulatore	Modalità	Versione di Android	Input	Esito
Test Case2_1	Inserimento nuovo valore di glucosio e nuovo valore di insulina in un determinato giorno ed ora.	WVGA (Nexus S) - Smartphone	Portrait Lingua Italiana	API 23	1. Premere sul cerchietto arancione item_glucose_status; 2. Premere su MODIFICA INSULINE; 3. Cliccare sul simbolo dell'ingranaggio; 4. Premere sul simbolo "+"; 5. Scrivere INIEZIONE5 nell'insulin_edit_name ; 6. Cliccare su EXTRA; 7. Cliccare su CENA; 8. Switchare BASALE su ON. 9. Switchare MEZZE UNITA' su ON e poi su OFF. 10. Cliccare SALVA; 11. Premere sul back button; 12. Cliccare su MODIFICA INSULINE; 13. Aggiungi il valore 12; 14. Premere su APPLICA; 15. Premere X; 16. Premere sul bottone blu AGGIUNGI NUOVO VALORE; 17. Cliccare due volte 1; 18. Cliccare 0; 19. Cliccare sull'editor_time; 20. Cliccare su ALTRA DATA; 21. Cliccare su 20 AGOSTO 2020; 22. Cliccare OK; 23. Cliccare 20; 24. Cliccare 32; 25. Cliccare OK; 26. Cliccare sulla spunta verde dell'editor_fab; 27. Cliccare sulla X.	Il Replay coincide con la fase di Capture.
Test Case2_2	Inserimento nuovo valore di glucosio e nuovo valore di insulina in un determinato giorno ed ora.	WVGA (Nexus S) - Smartphone	Portrait Lingua Inglese	API 23	1. Premere sul cerchietto arancione item_glucose_status; 2. Premere su MODIFICA INSULINE; 3. Cliccare sul simbolo dell'ingranaggio; 4. Premere sul simbolo "+"; 5. Scrivere INIEZIONE5 nell'insulin_edit_name ; 6. Cliccare su EXTRA; 7. Cliccare su CENA; 8. Switchare BASALE su ON. 9. Switchare MEZZE UNITA' su ON e poi su OFF. 10. Cliccare SALVA; 11. Premere sul back button; 12. Cliccare su MODIFICA INSULINE; 13. Aggiungi il valore 12; 14. Premere su APPLICA; 15. Premere X; 16. Premere sul bottone blu AGGIUNGI NUOVO VALORE; 17. Cliccare due volte 1; 18. Cliccare 0; 19. Cliccare sull'editor_time; 20. Cliccare su ALTRA DATA; 21. Cliccare su 20 AGOSTO 2020; 22. Cliccare OK; 23. Cliccare 20; 24. Cliccare 32; 25. Cliccare OK; 26. Cliccare sulla spunta verde dell'editor_fab; 27. Cliccare sulla X.	Non viene identificato il bottone MODIFICA INSULINE
Test Case2_3	Inserimento nuovo valore di glucosio e nuovo valore di insulina in un determinato giorno ed ora.	WVGA (Nexus S) - Smartphone	Portrait Lingua Spagnola	API 23	1. Premere sul cerchietto arancione item_glucose_status; 2. Premere su MODIFICA INSULINE; 3. Cliccare sul simbolo dell'ingranaggio; 4. Premere sul simbolo "+"; 5. Scrivere INIEZIONE5 nell'insulin_edit_name ; 6. Cliccare su EXTRA; 7. Cliccare su CENA; 8. Switchare BASALE su ON. 9. Switchare MEZZE UNITA' su ON e poi su OFF. 10. Cliccare SALVA; 11. Premere sul back button; 12. Cliccare su MODIFICA INSULINE; 13. Aggiungi il valore 12; 14. Premere su APPLICA; 15. Premere X; 16. Premere sul bottone blu AGGIUNGI NUOVO VALORE; 17. Cliccare due volte 1; 18. Cliccare 0; 19. Cliccare sull'editor_time; 20. Cliccare su ALTRA DATA; 21. Cliccare su 20 AGOSTO 2020; 22. Cliccare OK; 23. Cliccare 20; 24. Cliccare 32; 25. Cliccare OK; 26. Cliccare sulla spunta verde dell'editor_fab; 27. Cliccare sulla X.	Non viene identificato il bottone MODIFICA INSULINE

Fig. 3.30 Test Case 2

3.4.3 Test Case 3

Il terzo scenario previsto consiste nell'inserire un certo valore di glucosio in un determinato giorno e ad una determinata ora e ripetere la medesima operazione una seconda volta. Il test è stato registrato su di un Tablet Nexus 10 con API 24, in modalità portrait. Il test mira a verificare la riproducibilità di uno scenario di test, registrato su di uno schermo di dimensioni di circa 10 pollici, quindi un tablet, su dispositivi il cui schermo è nettamente inferiore, intorno ai 5/6 pollici, come uno smartphone. Gli step da eseguire sono i seguenti:

1. Premere sul bottone blu "Aggiungi nuovo valore";

2. Inserisci 1;
3. Inserisci 5;
4. Inserisci 9;
5. Premere sul bottone dell'editor_time;
6. Premere "seleziona un'altra data";
7. Seleziona 26 agosto 2020;
8. Premere "OK";
9. Seleziona 12;
10. Seleziona 50;
11. Premere "OK";
12. Premere sulla spunta verde;
13. Premere sulla "X";
14. Premere sul bottone blu "Aggiungi nuovo valore";
15. Inserisci 7;
16. Inserisci 5;
17. Inserisci 5;
18. Premere sul bottone dell'editor_time;
19. Selezionare "Today";
20. Seleziona 9;
21. Seleziona 30;
22. Premere "OK";
23. Premere sulla spunta verde;
24. Seleziona la "X";

Di seguito, si riporta porzione del codice corrispondente alle azioni appena citate:

```
1.  @Test
2.    public void sampleTest() throws InterruptedException{
3.      MobileElement el1 = (MobileElement) driver.findElementById("it.diab:id/fab");
4.      el1.click();
5.      Thread.sleep(2000);
6.      MobileElement el2 = (MobileElement) driver.findElementById("it.diab:id/keyboard_key_1"
    );
7.      el2.click();
8.      MobileElement el3 = (MobileElement) driver.findElementById("it.diab:id/keyboard_key_5"
    );
9.      el3.click();
10.     MobileElement el4 = (MobileElement) driver.findElementById("it.diab:id/keyboard_key_9"
    );
11.     el4.click();
12.     Thread.sleep(2000);
```

```
13.     MobileElement el5 = (MobileElement) driver.findElementById("it.diab:id/editor_time");

14.     el5.click();
15.     MobileElement el6 = (MobileElement) driver.findElementByXPath("/hierarchy/android.widg
    et.FrameLayout/android.widget.LinearLayout/android.widget.FrameLayout/android.widget.Frame
    Layout/android.widget.FrameLayout/androidx.appcompat.widget.LinearLayoutCompat/android.wid
    get.FrameLayout/android.widget.ListView/android.widget.TextView[3]");
16.     el6.click();
17.     MobileElement el7 = (MobileElement) driver.findElementByAccessibilityId("26 August 202
    0");
18.     el7.click();
19.     MobileElement el8 = (MobileElement) driver.findElementById("android:id/button1");
20.     el8.click();
21.     Thread.sleep(2000);
22.     MobileElement el9 = (MobileElement) driver.findElementByAccessibilityId("12");
23.     el9.click();
24.     MobileElement el10 = (MobileElement) driver.findElementByAccessibilityId("50");
25.     el10.click();
26.     Thread.sleep(2000);
27.     MobileElement el11 = (MobileElement) driver.findElementById("android:id/button1");
28.     el11.click();
29.     Thread.sleep(2000);
30.     MobileElement el12 = (MobileElement) driver.findElementById("it.diab:id/editor_fab");

31.     el12.click();
32.     MobileElement el13 = (MobileElement) driver.findElementByAccessibilityId("Close");
33.     el13.click();
34.     Thread.sleep(2000);
35.     MobileElement el14 = (MobileElement) driver.findElementById("it.diab:id/fab");
36.     el14.click();
37.     Thread.sleep(2000);
38.     MobileElement el15 = (MobileElement) driver.findElementById("it.diab:id/keyboard_key_7
    ");
39.     el15.click();
40.     MobileElement el16 = (MobileElement) driver.findElementById("it.diab:id/keyboard_key_5
    ");
41.     el16.click();
42.     el16.click();
43.     Thread.sleep(2000);
44.     MobileElement el17 = (MobileElement) driver.findElementById("it.diab:id/editor_time");

45.     el17.click();
46.     MobileElement el18 = (MobileElement) driver.findElementByXPath("/hierarchy/android.wid
    get.FrameLayout/android.widget.LinearLayout/android.widget.FrameLayout/android.widget.Fram
    eLayout/android.widget.FrameLayout/androidx.appcompat.widget.LinearLayoutCompat/android.wi
    dget.FrameLayout/android.widget.ListView/android.widget.TextView[1]");
47.     el18.click();
48.     MobileElement el19 = (MobileElement) driver.findElementByAccessibilityId("9");
49.     el19.click();
50.     MobileElement el20 = (MobileElement) driver.findElementByAccessibilityId("30");
51.     el20.click();
52.     Thread.sleep(2000);
53.     MobileElement el21 = (MobileElement) driver.findElementById("android:id/button1");
54.     el21.click();
55.     MobileElement el22 = (MobileElement) driver.findElementById("it.diab:id/editor_fab");

56.     el22.click();
57.     MobileElement el23 = (MobileElement) driver.findElementByAccessibilityId("Close");
58.     el23.click();
59.   }
```

Fig. 3.31 Codice corrispondente alle azioni previste per il Test Case 3

Il test è stato poi rieseguito sui seguenti dispositivi:

- WVGA (Nexus S) con API 26, device diverso (smartphone), in modalità portrait: il replay viene eseguito correttamente;
- Galaxy Nexus con API 23, device diverso (smartphone), in modalità portrait: il replay viene eseguito correttamente;

A valle del test, dunque, possiamo asserire che il test basato sullo scenario 3 viene registrato su di un Tablet, quindi un dispositivo il cui schermo si aggira intorno ai 10 pollici, e rieseguito correttamente su due smartphone differenti, quindi dispositivi il cui schermo risulta essere più piccolo (ad esempio, 4 o 5 pollici). C'è da sottolineare che, talvolta, il replay fallisce in quanto al momento dell'inserimento del valore di glucosio, la keyboard non compare e, conseguentemente, il test fallisce.

ID	Descrizione	Caratteristiche Emulatore	Modalità	Versione di Android	Input	Esito
Test Case3_1	Inserimento nuovo valore di glucosio in un determinato giorno ed ora per due volte.	Nexus 10 - Tablet	Portrait	API 24	1. Premere sul bottone blu AGIGUNGI NUOVO VALORE; 2. Inserire in successione 1, 5, 9; 3. Premere sul bottone dell'editor_time; 4. Premere SELEZIONA UN'ALTRA DATA; 5. Seleziona 26 agosto 2020 ; 6. Cliccare OK; 7. Premere sulla spunta verde; 8. Premere X; 9. Premere sul bottone blu AGGIUNGI NUOVO VALORE; 10. Inserisci in successione 7, 5, 5; 11. Premere sul bottone dell'editor_time; 12. Seleziona TODAY; 13. Seleziona 9; 14. Seleziona 30; 15. Premere OK ; 16. Premere sulla spunta verde; 17. Premere la X;	Il Replay coincide con la fase di Capture
Test Case3_2	Inserimento nuovo valore di glucosio in un determinato giorno ed ora per due volte.	WVGA (Nexus S) - Smartphone	Portrait	API 26	1. Premere sul bottone blu AGIGUNGI NUOVO VALORE; 2. Inserire in successione 1, 5, 9; 3. Premere sul bottone dell'editor_time; 4. Premere SELEZIONA UN'ALTRA DATA; 5. Seleziona 26 agosto 2020 ; 6. Cliccare OK; 7. Premere sulla spunta verde; 8. Premere X; 9. Premere sul bottone blu AGGIUNGI NUOVO VALORE; 10. Inserisci in successione 7, 5, 5; 11. Premere sul bottone dell'editor_time; 12. Seleziona TODAY; 13. Seleziona 9; 14. Seleziona 30; 15. Premere OK ; 16. Premere sulla spunta verde; 17. Premere la X;	
Test Case3_3	Inserimento nuovo valore di glucosio in un determinato giorno ed ora per due volte.	Galaxy Nexus - Smartphone	Portrait	API 23	1. Premere sul bottone blu AGIGUNGI NUOVO VALORE; 2. Inserire in successione 1, 5, 9; 3. Premere sul bottone dell'editor_time; 4. Premere SELEZIONA UN'ALTRA DATA; 5. Seleziona 26 agosto 2020 ; 6. Cliccare OK; 7. Premere sulla spunta verde; 8. Premere X; 9. Premere sul bottone blu AGGIUNGI NUOVO VALORE; 10. Inserisci in successione 7, 5, 5; 11. Premere sul bottone dell'editor_time; 12. Seleziona TODAY; 13. Seleziona 9; 14. Seleziona 30; 15. Premere OK ; 16. Premere sulla spunta verde; 17. Premere la X;	

Fig. 3.32 Test Case 3

3.4.4 Test Case 4

Il quarto scenario di test prevede l'inserimento di un nuovo valore di glucosio ad una certa data ed ora e, in aggiunta, prevede l'inserimento del valore dell'iniezione di insulina. Il test è stato registrato su un WVGA (Nexus) con API 23, in modalità portrait. Il test mira a verificare la riproducibilità del nostro scenario di test, registrato su un dispositivo il cui schermo si aggira intorno ai 4/5 pollici, quindi, uno smartphone, su dispositivi il cui schermo è nettamente maggiore, intorno ai 10 pollici, come nel caso del Tablet. Si tratta in effetti della procedura inversa previsa per Test Case 3. Gli step da eseguire sono i seguenti:

1. Premere sul bottone blu "Aggiungi nuovo valore";
2. Inserisci 7;
3. Inserisci 5;
4. Inserisci 3;
5. Premi sul bottone "editor_time";
6. Premere sul bottone "Seleziona un'altra data";
7. Seleziona 17 agosto 2020;
8. Seleziona "OK";
9. Seleziona 13;
10. Seleziona 25;
11. Premere "OK";
12. Premere sulla spunta dell'editor_fab;
13. Premere sulla "X";
14. Premere sul cerchietto arancione dell'item_glucose_status;
15. Premere su "Edit Insulin";
16. Inserire il valore 25;
17. Premere su "Apply";
18. Premere sul layout esterno;
19. Premere sulla "X";

Il codice corrispondente alle azioni pocanzi citate è il seguente:

```
1.  @Test
2.    public void sampleTest()throws InterruptedException {
3.      MobileElement el1 = (MobileElement) driver.findElementById("it.diab:id/fab");
4.      el1.click();
5.      Thread.sleep(2000);
6.      MobileElement el2 = (MobileElement) driver.findElementById("it.diab:id/keyboard_key_7"
    );
7.      el2.click();
```

```
8.      MobileElement el3 = (MobileElement) driver.findElementById("it.diab:id/keyboard_key_5"
    );
9.      el3.click();
10.     MobileElement el4 = (MobileElement) driver.findElementById("it.diab:id/keyboard_key_3"
    );
11.     el4.click();
12.     Thread.sleep(2000);
13.     MobileElement el5 = (MobileElement) driver.findElementById("it.diab:id/editor_time");

14.     el5.click();
15.     MobileElement el6 = (MobileElement) driver.findElementByXPath("/hierarchy/android.widg
    et.FrameLayout/android.widget.LinearLayout/android.widget.FrameLayout/android.widget.Frame
    Layout/android.widget.FrameLayout/androidx.appcompat.widget.LinearLayoutCompat/android.wid
    get.FrameLayout/android.widget.ListView/android.widget.TextView[3]");
16.     el6.click();
17.     MobileElement el7 = (MobileElement) driver.findElementByAccessibilityId("17 August 202
    0");
18.     el7.click();
19.     MobileElement el8 = (MobileElement) driver.findElementById("android:id/button1");
20.     el8.click();
21.     Thread.sleep(2000);
22.     MobileElement el9 = (MobileElement) driver.findElementByAccessibilityId("13");
23.     el9.click();
24.     MobileElement el10 = (MobileElement) driver.findElementByAccessibilityId("25");
25.     el10.click();
26.     MobileElement el11 = (MobileElement) driver.findElementById("android:id/button1");
27.     el11.click();
28.     Thread.sleep(2000);
29.     MobileElement el12 = (MobileElement) driver.findElementById("it.diab:id/editor_fab");

30.     el12.click();
31.     el12.click();
32.     MobileElement el13 = (MobileElement) driver.findElementByAccessibilityId("Close");
33.     el13.click();
34.     Thread.sleep(2000);
35.     MobileElement el14 = (MobileElement) driver.findElementById("it.diab:id/item_glucose_s
    tatus");
36.     el14.click();
37.     Thread.sleep(2000);
38.     MobileElement el15 = (MobileElement) driver.findElementById("it.diab:id/editor_insulin
    ");
39.     el15.click();
40.     Thread.sleep(2000);
41.     MobileElement el16 = (MobileElement) driver.findElementById("it.diab:id/glucose_editor
    _insulin_value");
42.     el16.sendKeys("25");
43.     el16.click();
44.     el16.click();
45.     MobileElement el17 = (MobileElement) driver.findElementById("it.diab:id/touch_outside"
    );
46.     el17.click();
47.     MobileElement el18 = (MobileElement) driver.findElementById("it.diab:id/editor_time");

48.     el18.click();
49.     Thread.sleep(2000);
50.     MobileElement el19 = (MobileElement) driver.findElementByAccessibilityId("Close");
51.     el19.click();
52.   }
```

Fig. 3.33 Codice corrispondente alle azioni previste per il Test Case 4.

Il test è stato poi rieseguito sui seguenti dispositivi:

- Nexus 10 con API 24, modalità portrait, device differente (tablet): il replay non viene eseguito correttamente. Il test fallisce allo step 14 in quanto non riesce a trovare il bottone identificato dall'item_glucose_status. Nella versione tablet dell'app, infatti, il bottone dell'item_glucose_status non è presente. In sostanza, la versione Tablet dell'app può essere considerata una versione riduttiva rispetto alla versione smartphone in quanto dà la possibilità di registrare un nuovo valore di glucosio, ma non un nuovo valore di iniezione di insulina. Dunque, la versione Tablet dell'app rappresenta una versione ridotta, rispetto alla duale versione smartphone, con funzionalità altrettanto limitate.
- Galaxy Nexus con API 23, modalità portrait, device differente: il replay viene eseguito correttamente.

A valle di questo di quest'ultimo scenario di test, dunque, possiamo asserire che il test registrato su un WVGA (Nexus S), smartphone, e rieseguito su di un Nexus 10, tablet, fallisce in quanto alcuni oggetti, presenti nell'app versione smartphone, non sono invece presenti nell'app versione tablet. Evidentemente, gli sviluppatori hanno previsto una versione dell'app soltanto smartphone. Conseguentemente, possiamo dedurre che la versione Tablet ne rappresenti una sua riduzione.

ID	Descrizione	Caratteristiche Emulatore	Modalità	Versione di Android	Input	Esito
Test Case4_1	Inserimento nuovo valore di glucosio, in un determinato giorno ed ora, ed inserimento del valore di insulina.	WVGA (Nexus S) - Smartphone	Portrait	API 23	1. Premere sul bottone blu AGIGUNGI NUOVO VALORE; 2. Inserire in successione 7, 5, 3; 3. Premere sul bottone dell'editor_time; 4. Premere SELEZIONA UN'ALTRA DATA; 5. Seleziona 17 agosto 2020 ; 6. Cliccare OK; 7. Seleziona 13; 8. Seleziona 25; 9. Premere OK; 10. Inserisci in successione 7, 5, 5; 11. Premere sulla spunta verde dell'editor_fab; 12. Premere X; 13. Premere sul cerchietto arancione dell'item_glucose_status; 14. Premere su EDIT_INSULIN; 15. Inserire il valore 25 ; 16. Premere APPLY; 17. Premere sul layout esterno; 18. Premere sulla X.	Il Replay coincide con la fase di Capture.
Test Case4_2	Inserimento nuovo valore di glucosio, in un determinato giorno ed ora, ed inserimento del valore di insulina.	Nexus 10 - Tablet	Portrait	API 24	1. Premere sul bottone blu AGIGUNGI NUOVO VALORE; 2. Inserire in successione 7, 5, 3; 3. Premere sul bottone dell'editor_time; 4. Premere SELEZIONA UN'ALTRA DATA; 5. Seleziona 17 agosto 2020 ; 6. Cliccare OK; 7. Seleziona 13; 8. Seleziona 25; 9. Premere OK; 10. Inserisci in successione 7, 5, 5; 11. Premere sulla spunta verde dell'editor_fab; 12. Premere X; 13. Premere sul cerchietto arancione dell'item_glucose_status; 14. Premere su EDIT_INSULIN; 15. Inserire il valore 25 ; 16. Premere APPLY; 17. Premere sul layout esterno; 18. Premere sulla X.	Non riesce ad identificare l'ITEM_GLUCOSE_STATUS
Test Case4_3	Inserimento nuovo valore di glucosio, in un determinato giorno ed ora, ed inserimento del valore di insulina.	Galaxy Nexus - Smartphone	Portrait	API 23	1. Premere sul bottone blu AGIGUNGI NUOVO VALORE; 2. Inserire in successione 7, 5, 3; 3. Premere sul bottone dell'editor_time; 4. Premere SELEZIONA UN'ALTRA DATA; 5. Seleziona 17 agosto 2020 ; 6. Cliccare OK; 7. Seleziona 13; 8. Seleziona 25; 9. Premere OK; 10. Inserisci in successione 7, 5, 5; 11. Premere sulla spunta verde dell'editor_fab; 12. Premere X; 13. Premere sul cerchietto arancione dell'item_glucose_status; 14. Premere su EDIT_INSULIN; 15. Inserire il valore 25 ; 16. Premere APPLY; 17. Premere sul layout esterno; 18. Premere sulla X.	Il Replay coincide con la fase di Capture.

Fig. 3.34 Test Case 4

3.4.5 Test Case 5

Il quinto scenario di test mira a verificare l'eliminazione di un valore di insulina all'interno di una lista di item. Il test è stato registrato su un WVGA (Nexus) con API 23, in modalità portrait. Gli step da eseguire sono i seguenti:

1. Premere sul cerchietto arancione dell'item_glucose_status;
2. Premere su ADD INSULIN;
3. Premere sul simbolo dell'ingranaggio;
4. Cliccare sul quinto elemento della lista;
5. Premere su REMOVE;
6. Switchare ALSO REMOVE ALL ITS VALUE FROM GLUCOSE READING su OFF;
7. Cliccare su CANCEL;

8. Cliccare su SAVE;
9. Premere BACK;
10. Premere APPLY;
11. Cliccare sulla X.

Di seguito, si riporta parte del codice corrispondente agli step appena citati:

```
1.  @Test
2.    public void sampleTest()throws InterruptedException {
3.      MobileElement el1 = (MobileElement) driver.findElementById("it.diab:id/item_glucose_st
    atus");
4.      el1.click();
5.      Thread.sleep(2000);
6.      MobileElement el2 = (MobileElement) driver.findElementById("it.diab:id/editor_insulin"
    );
7.      el2.click();
8.      Thread.sleep(2000);
9.      MobileElement el3 = (MobileElement) driver.findElementByAccessibilityId("Edit insulins
    ");
10.     el3.click();
11.     Thread.sleep(2000);
12.     MobileElement el4 = (MobileElement) driver.findElementByXPath("/hierarchy/android.widg
    et.FrameLayout/android.widget.LinearLayout/android.widget.FrameLayout/android.widget.Linea
    rLayout/android.widget.FrameLayout/android.view.ViewGroup/android.widget.FrameLayout/andro
    id.widget.FrameLayout/androidx.recyclerview.widget.RecyclerView/android.view.ViewGroup[5]/
    android.widget.TextView");
13.     el4.click();
14.     Thread.sleep(2000);
15.     MobileElement el5 = (MobileElement) driver.findElementById("it.diab:id/insulin_edit_de
    lete");
16.     el5.click();
17.     Thread.sleep(2000);
18.     MobileElement el6 = (MobileElement) driver.findElementById("it.diab:id/insulin_delete_
    dialog_values");
19.     el6.click();
20.     Thread.sleep(2000);
21.     MobileElement el7 = (MobileElement) driver.findElementById("android:id/button2");
22.     el7.click();
23.     Thread.sleep(2000);
24.     MobileElement el8 = (MobileElement) driver.findElementById("it.diab:id/insulin_edit_sa
    ve");
25.     el8.click();
26.     Thread.sleep(2000);
27.     MobileElement el9 = (MobileElement) driver.findElementByXPath("/hierarchy/android.widg
    et.FrameLayout/android.widget.LinearLayout/android.widget.FrameLayout/android.widget.Linea
    rLayout/android.widget.FrameLayout/android.view.ViewGroup/android.view.ViewGroup/android.w
    idget.ImageButton");
28.     el9.click();
29.     Thread.sleep(2000);
30.     MobileElement el10 = (MobileElement) driver.findElementById("it.diab:id/glucose_editor
    _insulin_negative");
31.     el10.click();
32.     Thread.sleep(2000);
33.     MobileElement el11 = (MobileElement) driver.findElementByAccessibilityId("Close");
34.     el11.click();
35.   }
```

Fig. 3.35 Codice corrispondente alle azioni previste per il Test Case 5.

Il test è stato poi rieseguito sui seguenti dispositivi:

- WVGA (Nexus S) con API 23, modalità portrait, stesso device: il replay viene eseguito correttamente.
- WVGA (Nexus S) con API 26, modalità portrait, stesso device: il replay viene eseguito correttamente;
- Nexus 10 con API 24, modalità portrait, device differente (Tablet): il replay viene eseguito correttamente.

ID	Descrizione	Caratteristiche Emulatore	Modalità	Versione di Android	Input	Esito
Test Case5_1	Eliminazione di un valore di insulina all'interno di una lista di item	WVGA (Nexus S) - Smartphone	Portrait	API 23	1. Premere sul cerchietto arancione dell'item_glucose_status; 2. Inserire ADD INSULIN; 3. Premere sul simbolo dell'ingranaggio; 4. Cliccare sul quinto elemento della lista; 5. Premere su REMOVE; 6. Switchare ALSO REMOVE ALL ITS VALUE FROM GLUCOSE READING su OFF; 7. Cliccare su CANCEL; 8. Cliccare su SAVE; 9. Premere BACK; 10. Premere APPLY; 11. Cliccare sulla X;	Il Replay coincide con la fase di Capture.
Test Case5_2	Eliminazione di un valore di insulina all'interno di una lista di item	WVGA (Nexus S) - Smartphone	Portrait	API 26	1. Premere sul cerchietto arancione dell'item_glucose_status; 2. Inserire ADD INSULIN; 3. Premere sul simbolo dell'ingranaggio; 4. Cliccare sul quinto elemento della lista; 5. Premere su REMOVE; 6. Switchare ALSO REMOVE ALL ITS VALUE FROM GLUCOSE READING su OFF; 7. Cliccare su CANCEL; 8. Cliccare su SAVE; 9. Premere BACK; 10. Premere APPLY; 11. Cliccare sulla X;	Il Replay coincide con la fase di Capture.
Test Case5_3	Eliminazione di un valore di insulina all'interno di una lista di item	Nexus 10 - Tablet	Portrait	API 24	1. Premere sul cerchietto arancione dell'item_glucose_status; 2. Inserire ADD INSULIN; 3. Premere sul simbolo dell'ingranaggio; 4. Cliccare sul quinto elemento della lista; 5. Premere su REMOVE; 6. Switchare ALSO REMOVE ALL ITS VALUE FROM GLUCOSE READING su OFF; 7. Cliccare su CANCEL; 8. Cliccare su SAVE; 9. Premere BACK; 10. Premere APPLY; 11. Cliccare sulla X;	Il Replay coincide con la fase di Capture.

Fig. 3.36 Test Case 5

3.4.6 Test Case 6

Il sesto ed ultimo scenario di test che prevediamo mira a verificare la possibilità di modificare il colore dello sfondo dell'applicazione e le soglie di glucosio, tramite delle seekbar, direttamente dal menù laterale. Il test è stato registrato su un WVGA (Nexus S) con API 26, in modalità portrait. Gli step da eseguire sono i seguenti:

1. Premere sui tre puntini corrispondenti al menù laterale;
2. Cliccare su STYLE;
3. Cliccare su DARK;
4. Cliccare su HIGH THRESHOLDS;
5. Cliccare su LOW THRESHOLDS;
6. Cliccare sul BACK BUTTON.

Di seguito, porzione del codice corrispondente alle azioni pocanzi descritte:

```
1.  @Test
2.    public void sampleTest()throws InterruptedException {
3.        MobileElement el1 = (MobileElement) driver.findElementByAccessibilityId("Settings");

4.          el1.click();
5.          Thread.sleep(2000);
6.          MobileElement el2 = (MobileElement) driver.findElementByXPath("/hierarchy/android.
    widget.FrameLayout/android.widget.LinearLayout/android.widget.FrameLayout/android.widget.L
    inearLayout/android.widget.FrameLayout/android.view.ViewGroup/android.widget.FrameLayout/a
    ndroid.widget.LinearLayout/android.widget.FrameLayout/androidx.recyclerview.widget.Recycle
    rView/android.widget.LinearLayout[1]/android.widget.RelativeLayout");
7.          el2.click();
8.          Thread.sleep(2000);
9.          MobileElement el3 = (MobileElement) driver.findElementByXPath("/hierarchy/android.
    widget.FrameLayout/android.widget.LinearLayout/android.widget.FrameLayout/android.widget.F
    rameLayout/android.widget.FrameLayout/androidx.appcompat.widget.LinearLayoutCompat/android
    .widget.FrameLayout/android.widget.ListView/android.widget.CheckedTextView[2]");
10.         el3.click();
11.         MobileElement el4 = (MobileElement) driver.findElementByXPath("/hierarchy/android.
    widget.FrameLayout/android.widget.LinearLayout/android.widget.FrameLayout/android.widget.L
    inearLayout/android.widget.FrameLayout/android.view.ViewGroup/android.widget.FrameLayout/a
    ndroid.widget.LinearLayout/android.widget.FrameLayout/androidx.recyclerview.widget.Recycle
    rView/android.widget.LinearLayout[5]/android.widget.LinearLayout[2]/android.widget.LinearL
    ayout/android.widget.SeekBar");
12.         el4.click();
13.         el4.click();
14.         MobileElement el5 = (MobileElement) driver.findElementByXPath("/hierarchy/android.
    widget.FrameLayout/android.widget.LinearLayout/android.widget.FrameLayout/android.widget.L
    inearLayout/android.widget.FrameLayout/android.view.ViewGroup/android.widget.FrameLayout/a
    ndroid.widget.LinearLayout/android.widget.FrameLayout/androidx.recyclerview.widget.Recycle
    rView/android.widget.LinearLayout[6]/android.widget.LinearLayout/android.widget.LinearLayo
    ut/android.widget.SeekBar");
15.         el5.click();
16.         Thread.sleep(2000);
17.         MobileElement el6 = (MobileElement) driver.findElementByXPath("/hierarchy/android.
    widget.FrameLayout/android.widget.LinearLayout/android.widget.FrameLayout/android.widget.L
    inearLayout/android.widget.FrameLayout/android.view.ViewGroup/android.view.ViewGroup/andro
    id.widget.ImageButton");
18.         el6.click();
19.         Thread.sleep(2000);
20.         MobileElement el7 = (MobileElement) driver.findElementByXPath("/hierarchy/android.
    widget.FrameLayout/android.widget.LinearLayout/android.widget.FrameLayout/android.widget.L
    inearLayout/android.widget.FrameLayout/android.view.ViewGroup/android.view.ViewGroup/andro
    id.widget.ImageButton");
21.         el7.click();
22.       }
```

Fig. 3.37 Codice corrispondente alle azioni previste per il Test Case 6.

Il test è stato poi rieseguito sui seguenti dispositivi:

- WVGA (Nexus S) con API 26, modalità portrait, stesso device: il replay viene eseguito correttamente.
- WVGA (Nexus S) con API 23, modalità portrait, stesso device: il replay viene eseguito correttamente;

- Nexus 10 con API 24, modalità portrait, device differente (Tablet): il replay viene eseguito correttamente.
- Galaxy Nexus con API 23, modalità portrait, device differente (vendor diverso): il replay viene eseguito correttamente.

ID	Descrizione	Caratteristiche Emulatore	Modalità	Versione di Android	Input	Esito
Test Case6_1	Modifica colore di layout e soglie di glucosio tramite seekbar	WVGA (Nexus S) - Smartphone	Portrait	API 26	1. Premere sui tre puntini corrispondenti al menù laterale; 2. Cliccare su STYLE; 3. Cliccare su DARK; 4. Cliccare su HIGH THRESHOLDS; 5. Cliccare su LOW THRESHOLDS; 6. Cliccare su BACK BUTTON.	Il Replay coincide con la fase di Capture.
Test Case6_2	Modifica colore di layout e soglie di glucosio tramite seekbar	WVGA (Nexus S) - Smartphone	Portrait	API 23	1. Premere sui tre puntini corrispondenti al menù laterale; 2. Cliccare su STYLE; 3. Cliccare su DARK; 4. Cliccare su HIGH THRESHOLDS; 5. Cliccare su LOW THRESHOLDS; 6. Cliccare su BACK BUTTON.	Il Replay coincide con la fase di Capture.
Test Case6_3	Modifica colore di layout e soglie di glucosio tramite seekbar	Nexus 10 - Tablet	Portrait	API 24	1. Premere sui tre puntini corrispondenti al menù laterale; 2. Cliccare su STYLE; 3. Cliccare su DARK; 4. Cliccare su HIGH THRESHOLDS; 5. Cliccare su LOW THRESHOLDS; 6. Cliccare su BACK BUTTON.	Il Replay coincide con la fase di Capture.
Test Case6_4	Modifica colore di layout e soglie di glucosio tramite seekbar	Galaxy Nexus - Smartphone	Portrait	API 23	1. Premere sui tre puntini corrispondenti al menù laterale; 2. Cliccare su STYLE; 3. Cliccare su DARK; 4. Cliccare su HIGH THRESHOLDS; 5. Cliccare su LOW THRESHOLDS; 6. Cliccare su BACK BUTTON.	Il Replay coincide con la fase di Capture.

Fig. 3.38 Test Case 6

.

3.5 APPLICAZIONE PER VEICOLI COMMERCIALI

Una tipica applicazione per veicoli commerciali consente al driver di controllare direttamente dal dispositivo mobile le funzionalità dell'abitacolo e del veicolo. Attraverso l'applicazione, infatti, è possibile accedere da remoto ai dati di diagnosi, ma anche avere una valutazione del proprio stile di guida, del consumo del carburante e così via. Inoltre, è possibile loggarsi all'interno dell'applicazione con le proprie credenziali o, in alternativa, entrare come guest, con funzionalità ridotte.

3.5.1 Test Case 1

Il primo scenario di test che prevediamo mira a verificare la possibilità di effettuare un login tramite le proprie credenziali che, conseguentemente, conferisce la possibilità di accedere pienamente a tutte le sezioni dell'applicazione. Il test registrato su un WVGA (Nexus S) con API 26, in modalità portrait, viene eseguito sullo stesso dispositivo più volte. Non sempre l'esito del test sarà positivo in quanto, dovendo staccare un token, la procedura sarà dipendente dalla suscettibilità del backend, che talvolta non è raggiungibile. Gli step da eseguire sono i seguenti:

1. Nel textbox dell'username, inserire la propria mail;
2. Nel textbox della password, inserire la propria password;
3. Premere su login.

Il test è stato poi rieseguito sui seguenti dispositivi:

- Nexus 10 con API 24, modalità portrait, device differente (tablet): il replay viene eseguito correttamente;
- WVGA (Nexus S) con API 26, modalità portrait, stesso device: il replay viene eseguito correttamente;
- Galaxy Nexus con API 26, modalità portrait, device diverso (vendor differente): il replay viene eseguito correttamente;

ID	Descrizione	Caratteristiche Emulatore	Modalità	Versione di Android	Input	Esito
Test Case1_1	Validazione della procedura di login.	Nexus 10 - Tablet	Portrait	API 24	1. Nel textbox dell'username, inserire la propria mail; 2. Nel textbox della password, inserire la propria password. 3. Premere su LOGIN;	Il Replay coincide con la fase di Capture.
Test Case1_2	Validazione della procedura di login.	WVGA (Nexus S) - Smartphone	Portrait	API 26	1. Nel textbox dell'username, inserire la propria mail; 2. Nel textbox della password, inserire la propria password. 3. Premere su LOGIN;	Il Replay coincide con la fase di Capture.
Test Case1_3	Validazione della procedura di login.	Galaxy Nexus - Smartphone	Portrait	API 26	1. Nel textbox dell'username, inserire la propria mail; 2. Nel textbox della password, inserire la propria password. 3. Premere su LOGIN;	Il Replay coincide con la fase di Capture.

Fig. 3.39 Test Case 1

3.5.2 Test Case 2

Il secondo scenario di utilizzo, invece, mira a verificare la possibilità di utilizzare l'applicazione, non attraverso la procedura del login e, quindi, con l'inserimento delle credenziali, bensì attraverso la funzionalità del login as guest, con funzionalità ridotte. Il test è stato registrato su un WVGA (Nexus S) con API 26, in modalità portrait. Gli step da eseguire sono i seguenti:

1. Premere su CONTINUE AS GUEST;
2. L'utente accetta la normalità alla privacy – GDPR;
3. Premere sul bottone relativo alla sezione dello stile di guida;
4. Premere il back button;
5. Premere sul bottone relativo al consumo del carburante.
6. Premere il back button;
7. Premere sul bottone dell'abitacolo;
8. Premere il back button.

Il test è stato poi rieseguito sui seguenti dispositivi:

- Nexus 10 con API 24, modalità portrait, device differente (tablet): il replay viene eseguito correttamente;
- WVGA (Nexus S) con API 26, modalità portrait, stesso device: il replay viene eseguito correttamente;
- Galaxy Nexus con API 26, modalità portrait, device diverso (vendor differente): il replay viene eseguito correttamente;

ID	Descrizione	Caratteristiche Emulatore	Modalità	Versione di Android	Input	Esito
Test Case2_1	Validazione della procedura del continue as guest.	Nexus 10 - Tablet	Portrait	API 24	1. Premere su CONTINUE AS GUEST; 2. Acettare la normaliva privacy - GDPR; 3. Premere il bottone relativo alla sezione dello stile di guida; 4. Premere il back button; 5. Premere il bottone relativo alla sezione del consumo del carburante; 6. Premere il back button; 7. Premere sul bottone relativo alla sezione dell'abitacolo; 8. Premere il back button.	Il Replay coincide con la fase di Capture.
Test Case2_2	Validazione della procedura del continue as guest.	WVGA (Nexus S) - Smartphone	Portrait	API 26	1. Premere su CONTINUE AS GUEST; 2. Acettare la normaliva privacy - GDPR; 3. Premere il bottone relativo alla sezione dello stile di guida; 4. Premere il back button; 5. Premere il bottone relativo alla sezione del consumo del carburante; 6. Premere il back button; 7. Premere sul bottone relativo alla sezione dell'abitacolo; 8. Premere il back button.	Il Replay coincide con la fase di Capture.
Test Case2_3	Validazione della procedura del continue as guest.	Galaxy Nexus - Smartphone	Portrait	API 26	1. Premere su CONTINUE AS GUEST; 2. Acettare la normaliva privacy - GDPR; 3. Premere il bottone relativo alla sezione dello stile di guida; 4. Premere il back button; 5. Premere il bottone relativo alla sezione del consumo del carburante; 6. Premere il back button; 7. Premere sul bottone relativo alla sezione dell'abitacolo; 8. Premere il back button.	Il Replay coincide con la fase di Capture.

Fig. 3.40 Test Case 2

3.5.3 Test Case 3

Il terzo ed ultimo scenario di utilizzo ha come finalità quella di verificare la possibilità che l'utente riesca ad eseguire correttamente la procedura di registrazione che gli consentirà di ottenere le credenziali necessarie per poter effettuare il login. Tuttavia, in questo scenario di test siamo stati costretti a rinunciare parzialmente alla fase di Capture (in quanto per poter completare correttamente tutti i campi relativi alla pagina della registrazione, la stessa doveva essere scrollata) con l'aggiunta di stringhe di codice che ci consentano di riprodurre lo scroll. Una delle limitazioni, infatti, di Appium consiste nel fatto che non consenta di scrollare una determinata pagina. Issue che al momento è stata aperta dagli utilizzatori e che, fino al momento della stesura della tesi, ancora non è stata fixata. Si immagina che il fix sarà compreso nella prossima release del tool. Tra l'altro, non è remota, infatti, la possibilità che ci si trovi di fronte alla necessità di una pagina che deve essere scrollata, dunque, il problema è tutt'altro che banale. Il test è stato registrato su un WVGA (Nexus S) con API 26, in modalità portrait. Gli step da eseguire sono i seguenti:

- Cliccare sul bottone REGISTER;
- Inserire il nome all'interno del campo NAME;
- Inserire il cognome all'interno del campo SURNAME;
- Inserire la mail nel campo EMAIL;
- Inserire il paese all'interno del campo COUNTRY;
- Inserire il prefisso all'interno del campo PREFIX;
- Inserire il numero di cellulare all'interno del campo PHONE;
- Inserire il nome dell'azienda all'interno del campo COMPANY;
- Inserire la password all'interno del campo PASSWORD;
- Riscrivere nuovamente la password all'interno del campo REPEAT PASSWORD;
- Spuntare la casella relativa alla privacy;
- Cliccare su I AGREE;
- Cliccare su I AGREE;
- Premere su SUBMIT.

Il test è stato poi rieseguito sui seguenti dispositivi:

- WVGA (Nexus S) con API 24, modalità portrait, stesso device: il replay viene eseguito correttamente;

- WVGA (Nexus S) con API 26, modalità portrait, stesso device: il replay viene eseguito correttamente;
- Galaxy Nexus con API 26, modalità portrait, device diverso (vendor differente): il replay viene eseguito correttamente;
- Nexus 10 con API 24, modalità portrait, device differente (tablet): il replay viene eseguito correttamente.

La stringa di codice che ci consente di scrollare la pagina relativa alla registrazione è la seguente:

```
1.  MobileElement elementToClick = (MobileElement) driver
2.      .findElementByAndroidUIAutomator("new UiScrollable("
3.          + "new UiSelector().scrollable(true)).scrollIntoView("
4.          + "new UiSelector().textContains(\"prefix\"));");
5.          elementToClick.click();
```

ID	Descrizione	Caratteristiche Emulatore	Modalità	Versione di Android	Input	Esito
Test Case3_1	Validazione della procedura di registrazione	WVGA (Nexus S) - Smartphone	Portrait	API 26	1. Cliccare sul bottone REGISTER; 2. Inserire il nome all'interno del campo NAME; 3. Inserire il cognome all'interno del campo SURNAME; 4. Inserire la mail nel campo EMAIL; 5. Inserire il paese all'interno del campo COUNTRY; 6. Inserire il numero di cellulare all'interno del campo PHONE; 7. Inserire il nome dell'azienda all'interno del campo COMPANY; 8. Inserire la password all'interno del campo PASSWORD; 9. Riscrivere nuovamente la password all'interno del campo REPEAT PASSWORD; 10. Spuntare la casella relativa alla privacy; 11. Cliccare su I AGREE; 12. Cliccare su I AGREE; 13. Cliccare su SUBMIT.	Il Replay coincide con la fase di Capture.
Test Case3_2	Validazione della procedura di registrazione	WVGA (Nexus S) - Smartphone	Portrait	API 24	1. Cliccare sul bottone REGISTER; 2. Inserire il nome all'interno del campo NAME; 3. Inserire il cognome all'interno del campo SURNAME; 4. Inserire la mail nel campo EMAIL; 5. Inserire il paese all'interno del campo COUNTRY; 6. Inserire il numero di cellulare all'interno del campo PHONE; 7. Inserire il nome dell'azienda all'interno del campo COMPANY; 8. Inserire la password all'interno del campo PASSWORD; 9. Riscrivere nuovamente la password all'interno del campo REPEAT PASSWORD; 10. Spuntare la casella relativa alla privacy; 11. Cliccare su I AGREE; 12. Cliccare su I AGREE; 13. Cliccare su SUBMIT.	Il Replay coincide con la fase di Capture.
Test Case3_3	Validazione della procedura di registrazione	Galaxy Nexus - Smartphone	Portrait	API 26	1. Cliccare sul bottone REGISTER; 2. Inserire il nome all'interno del campo NAME; 3. Inserire il cognome all'interno del campo SURNAME; 4. Inserire la mail nel campo EMAIL; 5. Inserire il paese all'interno del campo COUNTRY; 6. Inserire il numero di cellulare all'interno del campo PHONE; 7. Inserire il nome dell'azienda all'interno del campo COMPANY; 8. Inserire la password all'interno del campo PASSWORD; 9. Riscrivere nuovamente la password all'interno del campo REPEAT PASSWORD; 10. Spuntare la casella relativa alla privacy; 11. Cliccare su I AGREE; 12. Cliccare su I AGREE; 13. Cliccare su SUBMIT.	Il Replay coincide con la fase di Capture.
Test Case3_4	Validazione della procedura di registrazione	Nexus 10 - Tablet	Portrait	API 24	1. Cliccare sul bottone REGISTER; 2. Inserire il nome all'interno del campo NAME; 3. Inserire il cognome all'interno del campo SURNAME; 4. Inserire la mail nel campo EMAIL; 5. Inserire il paese all'interno del campo COUNTRY; 6. Inserire il numero di cellulare all'interno del campo PHONE; 7. Inserire il nome dell'azienda all'interno del campo COMPANY; 8. Inserire la password all'interno del campo PASSWORD; 9. Riscrivere nuovamente la password all'interno del campo REPEAT PASSWORD; 10. Spuntare la casella relativa alla privacy; 11. Cliccare su I AGREE; 12. Cliccare su I AGREE; 13. Cliccare su SUBMIT.	Il Replay coincide con la fase di Capture.

Fig. 3.41 Test Case 3

4. Risultati ed analisi

L'obiettivo di questo capitolo è di valutare la bontà dello strumento, oggetto del nostro studio, per gli scenari di test che abbiamo previsto nel capitolo precedente, partendo dall'ipotesi principale di avere a disposizione il solo APK dell'AUT, senza quindi aver accesso al codice sorgente della stessa.

4.1 Munchlife

In Munchlife, il replay relativo al click dei pulsanti, nella sola modalità portrait, viene eseguito correttamente su tutti i dispositivi oggetto di studio. Per quanto riguarda, invece, lo scenario di test relativo al click dei pulsanti in modalità portrait/landscape, abbiamo potuto constatare come il replay di un test registrato su un vendor in modalità portrait, fallisca sul medesimo in modalità landscape. Tale discorso risulta essere valido per uno smartphone, il cui schermo è delle dimensioni di circa 5/6 pollici. Al contrario, su un Tablet, il cui schermo è di dimensioni nettamente superiori, il replay in modalità landscape viene eseguito correttamente. In particolare, nel nostro caso, il replay fallisce nel momento in cui il sistema deve identificare il simbolo GENDER, che in modalità landscape, differentemente dalla modalità portrait, non è presente. Tutto ciò è riconducibile al fatto che il layout portrait ha delle dimensioni ben definite, motivo per il quale su uno schermo piccolo (uno smartphone), e quindi non un Tablet, il simbolo GENDER esce fuori dalla zona visibile. Avvalendosi dell'ausilio dei log, è possibile comprendere che l'altezza di alcuni campi, quale quella con i numeri, è fissata in termini della dimensione dei caratteri del font che, di conseguenza, non riesce a diminuire al di sotto di un certo valore, consentendo al simbolo GENDER di rimanere nella zona visibile. Un altro scenario di test particolarmente interessante è quello relativo alla registrazione di un test su un dispositivo in lingua inglese ed il suo replay su in dispositivo in lingua italiana e spagnola. In tal caso, differentemente dalle altre due applicazioni esaminate (Trolly e DIAB), il replay viene eseguito correttamente su tutti i dispositivi oggetto di studio in quanto l'id degli oggetti non cambia a seconda della lingua settata sul dispositivo. Infine, per l'ultimo scenario di test, abbiamo dovuto rinunciare parzialmente alla fase di Capture ed inserire una stringa di codice per avere la possibilità di tornare alla home page dell'applicazione, dopo aver modificato le impostazioni della stessa tramite l'ausilio del menù laterale. Essendo l'applicazione piuttosto obsoleta, essa è stata customizzata per dispositivi Android con un sistema operativo anch'esso vetusto. Tali dispositivi Android erano caratterizzati dalla presenza del back button fisico, come si può ben ricordare. I dispositivi Android più moderni, invece, non prevedono più la presenza del back button fisico. A tal proposito, l'applicazione lanciata su vendor Android con sistemi operativi più aggiornati, non avendo più a disposizione il back button fisico, richiederebbe l'aggiunta di una stringa che simuli il back button stesso per poter tornare alla pagina principale dell'applicazione. Stringa di codice, che, però dovrà essere istanziata ogni volta. La

stringa di codice che ci consente di simulare il back button laddove non presente è la seguente: *driver.navigate().back()*. Stesso discorso, come vedremo, varrà anche per Trolly, applicazione anch'essa obsoleta e customizzata per dispositivi Android vetusti.

4.2 Trolly

Anche in Trolly, il replay relativo all'individuazione ed il click dei pulsanti viene eseguito correttamente su tutti i vendor oggetto di studio. Un altro scenario di test particolarmente interessante consiste nella registrazione di un test su un vendor in lingua italiana ed il replay dello stesso su un dispositivo in lingua inglese e spagnola. Il replay, differentemente da MunchLife, fallisce su tutti i vendor oggetto di studio. Ciò è riconducibile al fatto che gli oggetti vengono identificati con un id che cambia a seconda della lingua settata sul device. In particolare, i test falliscono in quanto i locatori che individuano gli elementi dipendono dagli stessi test scritti sull'interfaccia, che cambia con la lingua. Tale inconveniente, però, teoricamente potrebbe essere risolto. Una prima alternativa consisterebbe nell'indicare non direttamente il testo, ma il puntatore nel file delle risorse in modo tale che venga letto dinamicamente il valore, in quel momento, in funzione. Tale soluzione è attualmente applicabile, ad esempio, nell'attuale versione di Espresso Test Recorder. Tuttavia, non è applicabile al nostro caso nel quale abbiamo a disposizione il solo APK dell'AUT. Infatti, se si esegue la fase di Capture in assenza di codice sorgente, come nel nostro caso, tale alternativa non è perseguibile in quanto, a tempo di esecuzione la risorsa è visibile come valore e non come puntatore al file delle risorse. Una seconda alternativa, invece, prevede la possibilità di non utilizzare i testi nella query per localizzare gli elementi. Anche questa soluzione non risulta essere perseguibile in quanto le query che non li considerano sono molto meno potenti e potrebbero condurre i test al fallimento per numerose altre cause. Per quanto riguarda, invece, lo scenario di test che prevede l'inserimento di item nella lista in modalità portrait/landscape, il replay di un test registrato su un vendor in modalità portrait, viene eseguito correttamente sul medesimo vendor ma anche su vendor differenti in modalità landscape. Infine, per l'ultimo scenario di test, anche qui, come per MunchLife, rinunciamo parzialmente alla fase di Capture per poter tornare alla Home Page dell'applicazione, una volta modificate le impostazioni della stessa tramite l'ausilio del menù laterale. Con l'aggiunta della medesima stringa di codice, vista per Munchlife (*driver.navigate().back()*), che dovrà essere istanziata ogni volta, possiamo simulare il back button fisico laddove non presente.

4.3 DIAB

Per DIAB, il replay relativo all'aggiunta di un nuovo valore di glucosio, ad una determinata ora e data, e l'aggiunta di un nuovo valore di insulina, viene eseguito correttamente su tutti i vendor oggetto di studio. Per quanto riguarda lo scenario di test che prevede la riesecuzione di un caso di test, registrato su un vendor in lingua italiana, su altri vendor in lingua inglese e spagnola, anche qui, vale il medesimo discorso già effettuato per Trolly. Il replay fallisce su tutti i dispositivi oggetto di studio in quanto gli oggetti dell'applicazione vengono identificati con un id che ha un nome differente a seconda della lingua settata nelle impostazioni di sistema del device. In particolare, dai log, si può facilmente notare come l'oggetto non venga identificato e, di conseguenza, il test fallisce, in quanto il test è stato registrato su un dispositivo la cui lingua di sistema era diversa rispetto alla lingua di sistema del dispositivo sul quale viene rieseguito. Le alternative presenti, seppur applicabili su tool diversi da quello oggetto del nostro studio (Espresso Test Recorder), risulterebbero comunque non valide in quanto prevedono l'accesso al codice sorgente, ipotesi ben diversa da quella dalla quale siamo partiti. Interessante è, inoltre, la replicabilità dello scenario di test che prevede l'inserimento di un nuovo valore di glucosio, ad una determinata data ed ora, e l'aggiunta di un nuovo valore di insulina su un dispositivo Tablet. In particolare, il replay di un test, registrato su uno smartphone, quindi, un dispositivo le cui dimensioni si aggirano intorno ai 4/5 pollici, fallisce su un Tablet, dispositivo le cui dimensioni si aggirano intorno ai 10 pollici. In particolare, il test fallisce allo step relativo all'inserimento del nuovo valore di insulina, in quanto il bottone relativo a tale comando (item_glucose_status), presente sullo smartphone, non è invece presente nella versione Tablet dell'applicazione. Verosimilmente, siamo giunti alla conclusione che gli sviluppatori abbiano previsto la sola versione smartphone dell'applicazione e che, quindi, la versione Tablet, ne rappresenti una sua riduzione. Infine, il replay, relativo allo scenario di test che prevede l'inserimento di nuovo valore di glucosio, ad una certa data ed ora, registrato su un vendor in modalità portrait, non sempre conduce ad un esito positivo in modalità landscape. Ciò è causato dal fatto che talvolta, al momento dell'inserimento del valore di glucosio, la keyboard non compare e, conseguentemente, il test fallisce.

4.4 APPLICAZIONE PER VEICOLI COMMERCIALI

Il primo scenario di test che prevede l'inserimento delle proprie credenziali per effettuare il login e, quindi, aver accesso a tutte le sezioni dell'applicazione, non sempre ha un replay positivo. Ciò è legato al fatto che, essendo la procedura del login legata all'inserimento di un username ed una password, memorizzate all'interno del backend, l'esito del processo di autenticazione è legato alla disponibilità del backend stesso, non sempre raggiungibile. Talvolta, infatti, quando il server è giù,

la procedura di autenticazione fallisce e a video viene visualizzato un messaggio di errore del server. Il replay, invece, del secondo scenario di test che mira a validare la possibilità di utilizzare l'applicazione non attraverso la procedura del login, e quindi, con l'inserimento delle proprie credenziali, ma attraverso la procedura del *login as guest*, con funzionalità ridotte, viene eseguito correttamente su tutti i vendor oggetto di studio. Degno di nota, invece, è l'ultimo scenario di test previsto che consiste nel validare la procedura di registrazione che consentirà, poi, all'utente di ottenere le credenziali per effettuare il login all'interno dell'applicazione. Anche in questo caso, rinunceremo parzialmente alla fase di Capture. Per poter terminare con successo la procedura di registrazione, bisognerà completare tutti i text field all'interno della pagina stessa che risulta essere scrollabile. Tuttavia, una delle limitazioni significative di Appium consiste nel fatto che esso non consente di scrollare una determinata pagina. Tale issue è ben nota agli utilizzatori (il link della issue su github è la seguente → https://github.com/appium/appium/issues/14418) e che, sino al momento della stesura della tesi, ancora non è stata corretta. Verosimilmente, la prossima release del tool conterrà anche il fix di tale problematica. Tuttavia, non è remota la possibilità che ci si trovi di fronte ad una pagina scrollabile (come nel nostro caso) e per ovviare a tale inconveniente, rinunciamo parzialmente alla fase di Capture, come già detto, introducendo la seguente stringa di codice che ci consente di riprodurre lo scroll:

```
1.  MobileElement elementToClick = (MobileElement) driver
2.      .findElementByAndroidUIAutomator("new UiScrollable("
3.          + "new UiSelector().scrollable(true)).scrollIntoView("
4.          + "new UiSelector().textContains(\"prefix\"));");
5.          elementToClick.click();
```

Fig. 4.1 Stringa di codice per la simulazione dello scroll

5. Discussioni

Il seguente capitolo ha come finalità quella di effettuare un confronto generale sulla base dei test eseguiti, indipendentemente dalla singola applicazione oggetto di studio. A tal proposito, viene in aiuto la seguente tabella:

		WVGA (Nexus S) API 26	**WVGA (Nexus S) API 27**	**WVGA (Nexus S) API 28**	**Nexus 10 API 27**	**Nexus 10 API 24**	**WVGA (Nexus S) API 23**	**Galaxy Nexus API 23**	**Galaxy Nexus API 23**
Munchlife	Verifica della rieseguibilità dei test registrati	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6
	Portrait/Landscape	0/6	N.A.	N.A.	6/6	N.A.	0/6	N.A.	N.A.
	Language	6/6	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.
Trolly	Verifica della rieseguibilità dei test registrati	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6
	Portrait/Landscape	6/6	N.A.	N.A.	N.A.	6/6	N.A.	N.A.	N.A.
	Language	0/6	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.
DIAB	Verifica della rieseguibilità dei test registrati	6/6	6/6	6/6	0/6	0/6	6/6	6/6	6/6
	Portrait/Landscape	0/6	N.A.	N.A.	N.A.	N.A.	0/6	N.A.	N.A.
	Language	0/6	N.A.	N.A.	N.A.	N.A.	0/6	N.A.	N.A.
Applicazione per veicoli commerciali	Verifica della rieseguibilità dei test registrati	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6
	Portrait/Landscape	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.
	Language	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.

Fig. 5.1 Tabella riepilogativa delle tipologie di test per applicazione.

La precedente tabella riporta sulle righe le applicazioni, oggetto del nostro studio, e sulle colonne le caratteristiche degli emulatori sui quali sono stati registrati i casi di test. Tale tabella ci consente, in linea più generale, di avere una visione d'insieme circa le similitudini di alcuni casi verificatisi su applicazioni diverse. In particolare, possiamo riassumere le conclusioni, alle quali siamo giunti grazie alla nostra analisi, mediante le seguenti domande/risposte:

- **Quanto spesso i test di Appium sono rieseguibili?**
  Con la denominazione "*Verifica della rieseguibilità dei test registrati*", all'interno della tabella, abbiamo indicato il caso in cui l'output sia uguale a quello che si deve manifestare a seguito di una determinata funzionalità. La rieseguibilità, nelle stesse condizioni nelle quali è stato registrato il test, dà sempre esito positivo. Nella Fig. 4.2, come si può ben notare, il replay viene eseguito correttamente su tutti i vendor e su tutte le applicazioni, oggetto di studio, 6/6.

- **Quanto spesso i test di Appium sono replicabili e quali sono le cause di mancata replicabilità?**
  Una delle cause di mancata replicabilità, in condizioni leggermente diverse dalle quali è stato registrato il test è il seguente: Portrait/Landscape. Ciò è legato al fatto che il layout portrait ha delle dimensioni ben definite rispetto a quelle landscape. L'altezza di alcuni campi (come quelli dei numeri in Munchlife) è fissata in termini della dimensione dei caratteri del font che, di conseguenza, non riesce a diminuire al di sotto di un certo valore. Tra l'altro, se registrassimo la stessa tipologia di test in modalità landscape, il replay, nelle stesse modalità, verrebbe eseguito correttamente, proprio a sostegno delle nostre ipotesi. Un ulteriore tipologia di test, che ricade nelle cause di mancata replicabilità, è quella della lingua. Gli oggetti all'interno dell'applicazione vengono identificati con un id che ha un nome differente a seconda della lingua settata nelle impostazioni di sistema del device. Di conseguenza, il replay fallisce in tutte le applicazioni oggetto di studio, ad eccezione di Munchlife, dove gli oggetti sono identificati con un id che non cambia a seconda della lingua.

- **Quali sono le limitazioni che incontra il tester nell'utilizzo di Appium?**
  Le versioni di Appium possono non risultare compatibili con le versioni OS sia di Android che di iOS. Come si può facilmente notare, dalla nostra analisi sono stati esclusi device con OS Android 10. Sono stati, infatti, considerati device con versioni precedenti del sistema operativo. Con l'ultima versione di Appium, attualmente disponibile sul web (che coincide

con quella utilizzata nel lavoro di tesi) non è possibile, infatti, registrare nessun test su un dispositivo Android con versione 10. Il pop up mostrato risulta essere il seguente:

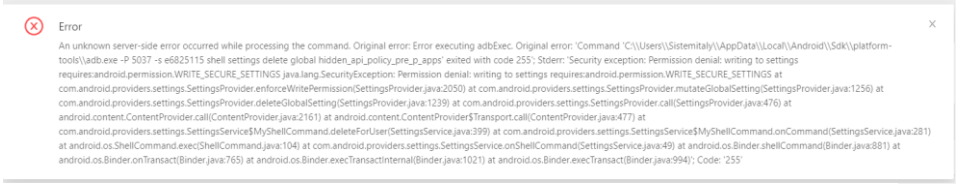


Fig. 5.2 Pop up di errore in Appium di Android 10

Presumibilmente, ciò è riconducibile al nuovo modello, più dettagliato, di autorizzazioni previsto per Android 10. Si tratta ovviamente di una problematica tediosa per l'attività di automazione per i test, con un gran vantaggio in termini di privacy. Evidentemente, Appium, sviluppato, come già detto, da Experitest e non da Google, è indietro con gli aggiornamenti verso Android 10. Differentemente, Espresso Test Recorder non possiede tale problematica. Essendo sviluppata da Google, è in grado di conoscere in anticipo tali aggiornamenti. Un'altra problematica, invece, consiste nell'assenza, all'interno di Appium, di alcune feature, come quella dello scroll. Funzionalità che abbiamo validato per la sola applicazione per veicoli commerciali. Pur non essendo presente nelle altre tre applicazioni studiate, non è infatti remota l'eventualità di trovarsi di fronte ad un'applicazione che abbia una pagina scrollabile. In tal caso, la registrazione e l'esecuzione del test con Appium risulta essere un po' più elaboriosa. in quanto tale tool non fornisce la possibilità di "catturare" lo scroll. Conseguentemente, il test si riterrà non registrato correttamente dal momento che bisognerà rinunciare parzialmente alla fase di Capture ed inserire una stringa di codice che consenta di simulare lo scroll e che dovrà essere istanziata ogni volta. Sul web, numerosi sono i suggerimenti che consentono di simulare lo scroll. Tuttavia, non è per nulla semplice l'individuazione della stringa che confà al proprio caso. Non è difficile, infatti, trovarsi di fronte ad istruzioni deprecate. Bisogna trovare un allineamento tra la versione di Appium che si sta utilizzando, la versione del java-client e quella di Selenium. Operazione non immediata e piuttosto tediosa. Questa tematica, come anche quella delle autorizzazioni, sarà evidentemente inclusa nella prossima release di Appium.

- **Quali sono i vantaggi di Appium rispetto ad altri strumenti simili?**
  È un tool che consente di automatizzare test per applicazioni mobili sia su piattaforma Android che iOS con il minimo effort. In particolare, il replay può avvenire in un IDE di scelta (nel nostro caso, Eclipse), nel linguaggio di programmazione desiderato (sempre nel nostro caso, JUnit). Appium si configura come un tool particolarmente semplice ed intuitivo: non richiede una conoscenza profonda di programmazione, quindi, anche tester con meno esperienza possono approcciarsi all'utilizzo di questo tool con estrema facilità, grazie anche ai numerosi tutorial messi a disposizione, sia dall'azienda sviluppatrice del tool stesso, Experitest, ma anche dagli sviluppatori che lo utilizzano. Si presta bene a situazioni, come la nostra, nelle quali non si ha accesso al codice sorgente, ma si dispone del solo APK dell'applicazione. Infine, non è necessario ricompilare o modificare l'applicazione oggetto di studio, grazie all'utilizzo di API di automazione standard su tutte le piattaforme.

- **Quanto Appium è adatto a risolvere i problemi di testing di applicazioni Android in ambito industriale?**
  In ambito industriale, con grande probabilità ci saranno feature più complesse e strutturate, come ad esempio, la necessità di effettuare la procedura del login. La procedura del login, nella nostra analisi, è stata validata solo per l'applicazione per veicoli commerciali, in quanto, nelle altre applicazioni, oggetto di studio, non vi è alcuna procedura che richieda l'autenticazione dell'utente. Il replay, in questo caso, non è sempre eseguito correttamente dal momento che si tratta di una tipologia di test che richiede il coinvolgimento del backend. Il successo o il fallimento della validazione della procedura di autenticazione sono, quindi, legati alla disponibilità del backend stesso. In caso contrario, non sarà possibile autenticarsi e, quindi. accedere alle funzionalità dell'applicazione. È importante, però, sottolineare come questa problematica, in realtà, sia una difficoltà intrinseca di questa tipologia di tecniche e non è imputabile alla scelta del tool, e, quindi, ad Appium. In aggiunta a tale problematica, vi è anche l'assenza di feature come quella dello scroll. Con grande probabilità, infatti, vi sarà la necessità di scrollare un'intera sezione dell'applicazione o parte di essa. In questo caso, come già citato precedentemente, non essendo implementata in Appium la possibilità di "catturare" lo scroll, sarà necessario ovviare a tale inconveniente inserendo una stringa di codice che simuli lo scroll, rinunciando parzialmente alla fase di Capture, e che dovrà essere istanziata ogni volta. A meno di questi casi appena citati, Appium si presta bene al testing di applicazioni Android in ambito industriale.

	Login	Scroll
Applicazione per veicoli commerciali	3/6	6/6
Trolly	Not available	Not available
DIAB	Not available	Not available
MunchLife	Not available	Not available

Fig. 5.3 Funzionalità di Login e Scroll

Punti di forza

- Open source
- Semplice ed intuitivo, grazie alla presenza di numerosi tutorial di supporto
- Non richiede una conoscenza di programmazione approfondita
- Consente di automatizzare test per applicazioni mobile sia su piattaforma Android che iOS riducendo l'effort che si avrebbe con l'esecuzione manuale
- Replay in un IDE di propria scelta utilizzando qualsiasi linguaggio compatibile con il WebDriver come Java, Python, C, ecc.
- Non è necessario il source code: è sufficiente l'APK
- Non è necessario ricompilare o modificare l'app grazie all'utilizzo di API di automazione standard su tutte le piattaforme.

Punti di debolezza

- Alcune versioni di Appium possono non risultare compatibili con le versioni OS di Android ed iOS
- Alcune feature non sono presenti (come lo scroll). Bisogna, quindi, rinunciare parzialmente alla fase di replay aggiungendo stringhe di codice che dovranno essere istanziate ogni volta

Bibliografia e siti web

[1] Mattia Fazzini, Eduardo Noronha de A. Freitas, Shauvik Roy Choudhary and Alessandro Orso. Barista: A Technique for Recording, Encoding and Running Platform Independent Android Tests. *In Software Testing, Verification and Validation (ICST), 2017 IEEE International Conference on. IEEE, 149-160.*

[2] Sergio Di Martino, Anna Rita Fasolino, Luigi Libero Lucio Starace, Porfirio Tramontana. Comparing the Effectiveness of Capture and Replay against Automatic Input Generation for Android GUI Testing.

[3] Lorenzo Gomez et al. "Reran: Timing- and touch-sensitive record and replay for android". *In: Proceedings of 2013 International Conference on Software Engineering, IEEE Press.2013, pp. 72-81.*

[4] Emil Alègroth, Zebao Gao, Rafael A.P. Oliviera, Atif Memon. Conceptualization and Evaluation of Component-based Testing Unified with Visual GUI Testing: an Empirical Study. *In Software Testing, Verification and Validation (ICST), 2015 IEEE 8th International Conference on. IEEE, 1-10.*

[5] Luca Ardito, Riccardo Coppola, Maurizio Morisio and Marco Torchiano. Espresso vs. Eyeautomate: An Experiment for the Comparison of Two Generations of Android GUI Testing.

[6] Matthew Halpern, Yuhao Zhu, Ramesh Peri, Vijay Janapa Reddi. Mosai: Cross-Platform User-Interaction Record and Replay for the Fragmented Android Ecosystem.

[7] Kevin Moran, Richard Bonett, Carlos Bernal-Càrdenas, Brendan Otten, Daniel Park & Denys Poshyvanyk. On-Device Bug Reporting For Android Applications. *In: 2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft).*

[8] Tom Yeh, Tsung-Hsiang Chang, and Robert C. Miller. 2009. Sikuli: using GUI screenshots for search and automation. *In Proceedings of the 22nd annual ACM symposium on User Interface software and technology (UIST '09). ACM, New York, NY, US, 183-192.*

[9] Onur Sahin, Assel Aliyeva, Hariharan Mathavan, Ayse Coskun, Manuel Egele. Late Breaking Results: Toward Practical Record and Replay For Mobile Applications.

[10] Porfirio Tramontana. Slide del corso di Ingegneria del Software 2. Testing: Fondamenti teorici.

[11] Porfirio Tramontana. Slide del corso di Ingegneria del Software 2. Android.

[12] Porfirio Tramontana. Slide del corso di Ingegneria del Software 2. Testing di unità: JUnit.

[13] Porfirio Tramontana. Slide del corso di Ingegneria del Software 2. Verifica e Validazione del Software: Testing Back box.

[14] Porfirio Tramontana. Slide del corso di Ingegneria del Software 2. Software Testing: User Interface Testing.

[15] Porfirio Tramontana. Slide del corso di Ingegneria del Software 2. Android Testing.

[16] Porfirio Tramontana. Slide del corso di Ingegneria del Software 2. User Session Based Testing.

[17] Diab: https://github.com/bvlj/diab

Commentato [PT6]: La bibliografia è completa però dovresti collegarla al testo: nel punto del testo in cui parli, ad esempio di Reran dovrai scrivere [3] per indicare a chi legge di consultare il riferimento bibliografico [3] per approfondimenti.

[18] https://docs.experitest.com/display/TC/AS+-+Getting+started

[19] https://docs.experitest.com/display/TC/AS+-+Android+-+Build+your+first+test

[20] https://docs.experitest.com/display/TC/AS+-+Run+your+existing+Appium+tests

[21] Appium: http://appium.io/

[22] Espresso Test Recorder: https://developer.android.com/studio/test/espresso-test-recorder

[23] Installazione di git: https://www.html.it/pag/53180/installazione-di-git/

[24] Integrare github in Android Studio: https://www.html.it/pag/67712/integrare-github-in-android-studio/