***Elaborato di Ingegneria del Software 2***

***Galli Antonio M63000721***

***Gravina Michela M63000708***

***Valletta Paolo M63000723***

1. ***Parte 0: Guida alla struttura della repo***

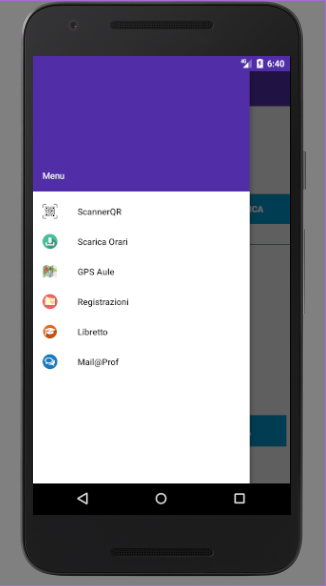
La repo contiene tutto il materiale realizzato e adoperato per il progetto in essere. La root principale contiene, oltre i file del progetto Android Studio dell’app UninaCampus, una directory “Tesina”. Al suo interno troviamo:

* ApplicazioniTestate: tutte le applicazioni usate per riconoscere le differenze nei test Espresso, Robotium e Robolectric, e da cui sono stati effettuati i test di traduzione automatica. Nello specifico, troviamo “MunchLife”, “Trolly” e “appTest” (applicazione demo comprendente solo elementi grafici, usata come base di partenza per trovare le diverse modalità operative dei tre strumenti di test prima enunciat);
* TraduttoriAutomatici: contenenti i progetti Java dei programmi realizzati per eseguire la traduzione dei test da Espresso a Robolectric;
* ClassDiagram.vpp: file comprendente il diagramma delle classi dell’app Android realizzata;
* MappaGenerica.xlsx e mappeNuove.xlsx: i file comprendenti le associazioni e traduzioni in termini di istruzioni tra Espresso, Robotium e Robolectric trovate;
* ProvaTest\_annotato.java, ProvaTest2\_annotato.java e ProvaTes3t\_annotato.java: i file realizzati con i tool automatici, che di seguito saranno citati e analizzati;
* TesinaIS2GalliGravinaValletta.docx: il presente file.

1. ***Parte 1: Unina Campus***
   1. ***Requisiti***

Unina Campus è una applicazione Android che permette ad uno studente che la utilizza di gestire la propria carriera universitaria. Offre le seguenti funzionalità:

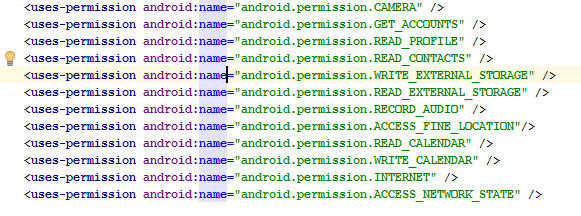
* **ScannerQR:** uno studente può inquadrare il codice QR posto fuori ogni aula al fine di vedere, nel giorno corrente, in quali ore e da quali lezioni l’aula è occupata.
* **Selezione corsi da seguire:** lo studente può selezionare i corsi del suo Corso di laurea da seguire.
* **Scarica Orari:** lo studente può scaricare gli orari dei corsi che ha scelto di seguire. Tali orari verranno sincronizzati nel calendario del dispositivo. È necessario, quindi, avere un calendario sul dispositivo per poter usufruire di tale funzionalità.
* **GPS Aule:** lo studente può visualizzare sulla mappa la posizione propria e di un’aula scelta.
* **Gestione Registrazioni:** lo studente può effettuare l’upload della registrazione di una lezione, oppure effettuare il download delle registrazioni caricate da altri studenti.
* **Libretto:** l’applicazione tiene traccia degli esami sostenuti dallo studente, il quale può aggiornare la sua carriera. È possibile aggiungere un esame o eliminarlo. In entrambi casi verrà aggiornata la media dello studente e di conseguenza la lista dei corsi che può seguire.
* **Mail@Prof:** Lo studente può inviare un’email ad un professore selezionando un corso dalla lista dei corsi da lui seguiti, oppure specificando l’email di destinazione. In entrambi i casi l’invio dell’email viene demandata ad un client esterno.
* **Registra lezione:** lo studente può selezionare un corso e quindi registrare la lezione. Il file viene memorizzato sulla memoria del dispositivo.



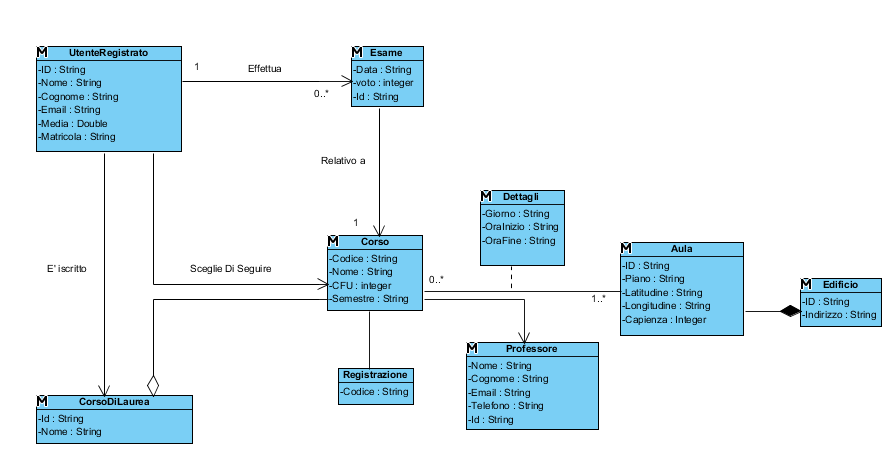
* 1. ***Struttura dell’applicazione***

L’applicazione consta sostanzialmente di 3 package UI, Business, Entity. Il primo è costituito dalle activity e ha lo scopo di interfacciarsi con l’utente e di cogliere tutti gli eventi esterni. Il secondo package è responsabile della logica dell’applicazione. Il package Entity, infine, contiene le classi che rappresentano le entità del dominio dell’applicazione. Ha lo scopo di mappare le tabelle del database in oggetti java.

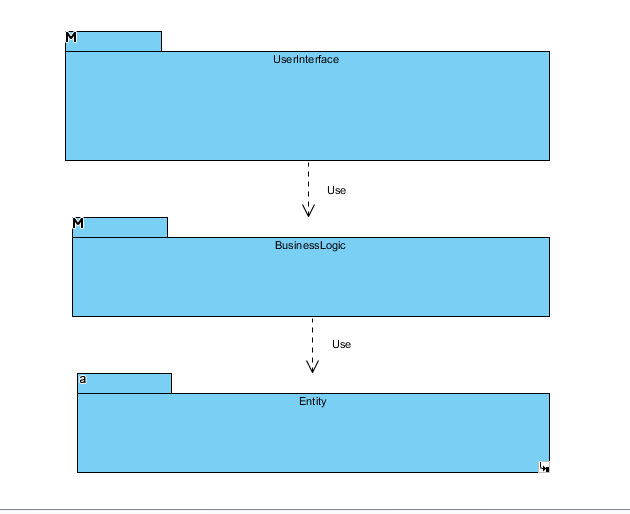
Per il corretto funzionamento, l’applicazione necessita dei seguenti permessi, riportati nel Manifest:



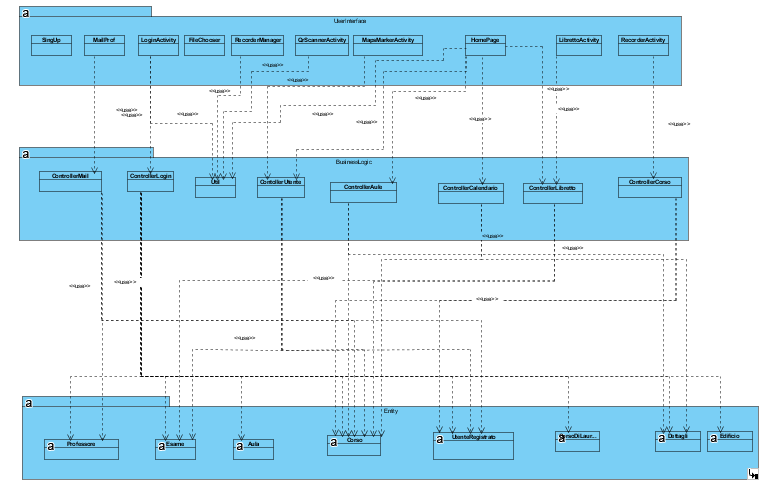
*Class Diagram Entity.*



*Package Diagram Generico*



*Package Diagram nel dettaglio*

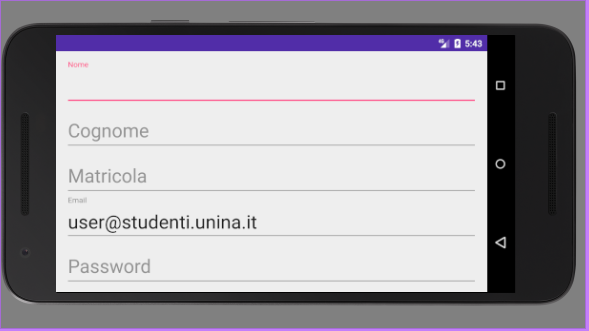


Per questioni di visibilità si rimanda al progetto in Visual Paradigm.

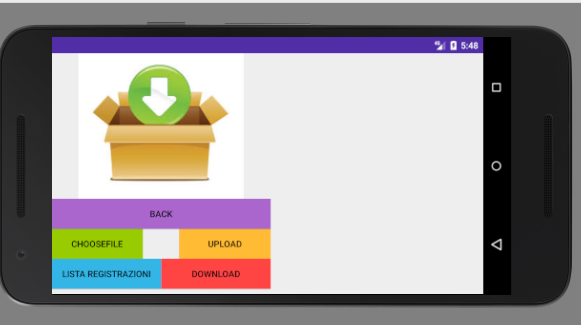
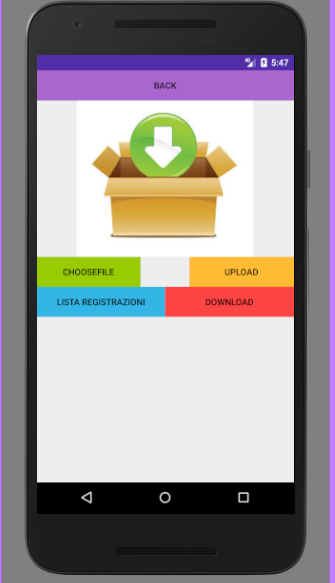
Di seguito è riportata un’analisi delle classi dell’applicazione.

* ***Package User Interface:***

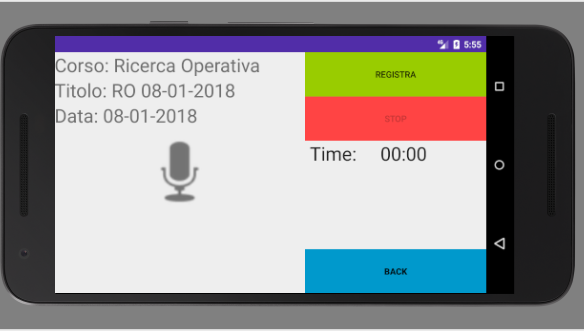
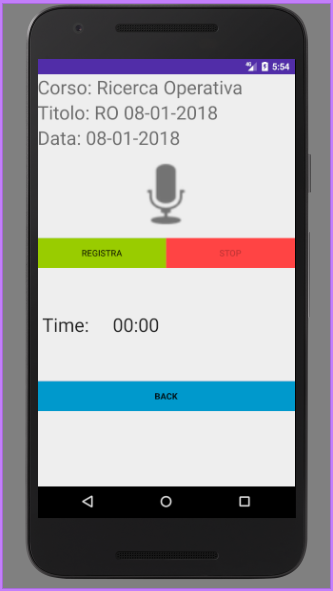
**SingUp:** Tale classe permette ad utente di immettere i dati per registrarsi e quindi utilizzare l’applicazione. Tramite il metodo attemptSignUp() viene effettuata la registrazione, usufruendo delle funzionalità di Firebase, mentre con il metodo isValidEmail() viene controllata la validità dell’email con cui l’utente ha deciso di iscriversi; in particolare è possibile usare solo una email del dominio @studenti.unina.it.



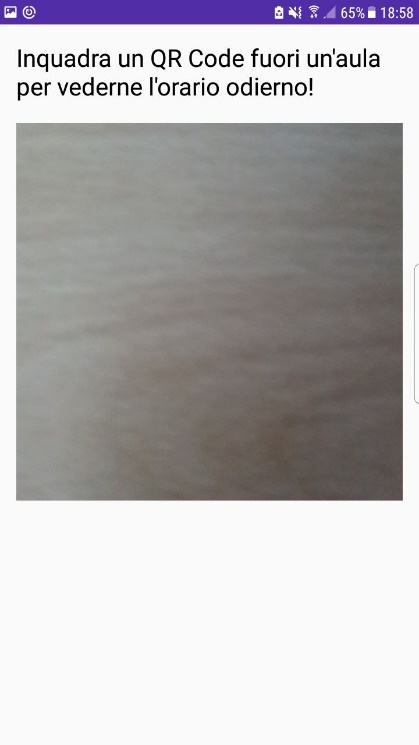
* + **RecorderManager:** Tale classe permette all’utente di scegliere una registrazione da caricare, oppure vedere la lista delle registrazioni presente online ed eventualmente di selezionarne una per cui effettuare un download. Le funzionalità di tale classe accedono alla memoria esterna del dispositivo e pertanto sono necessari gli opportuni permessi, richiesti dalla funzione requestPermission() e verificati dal metodo checkPermission(). Inoltre la classe invoca le funzionalità della classe FileChooser.



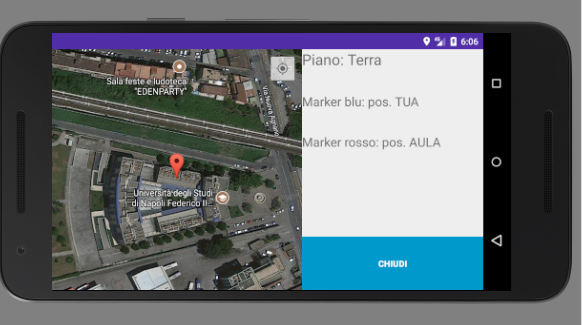
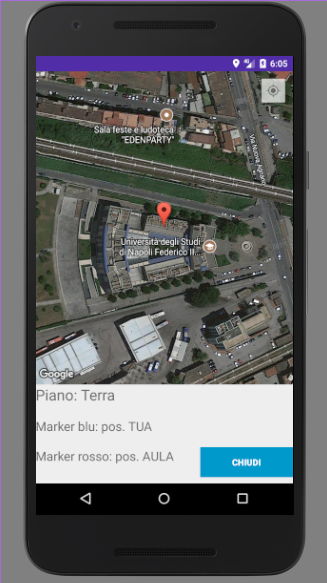
* + **RecorderActivity:** Tale classe è responsabile di effettuare una registrazione, quindi richiede i permessi per accedere al microfono con la funzione requestPermission(). Alla fine della registrazione viene generato un file in formato “acc” che viene salvato nella memoria esterna del dispositivo.



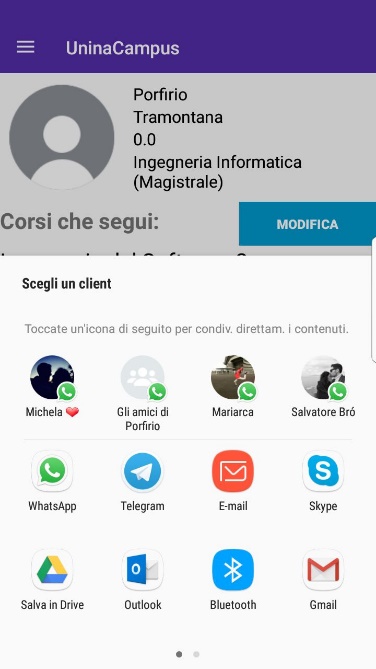
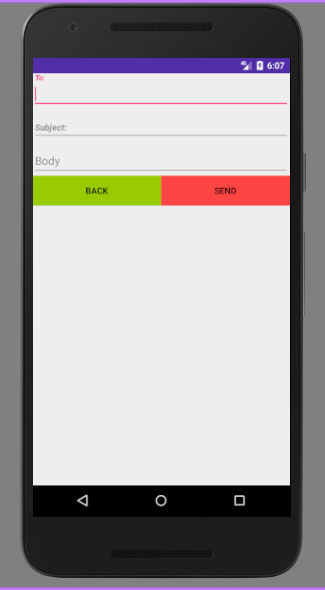
* + **QRScannerActivity:** Tale classe è responsabile di leggere il codice QR presente fuori ogni classe, quindi richiede i permessi per accedere alla fotocamera.



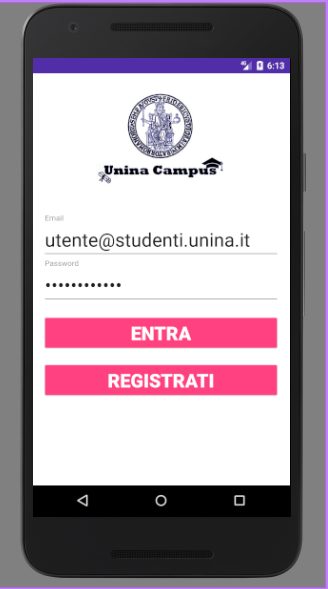
* + **MapsMarkerActivity:** Tale classe ha il compito di mostrare la posizione dell’utente e dell’aula. Necessita dei permessi per accedere alla localizzazione.



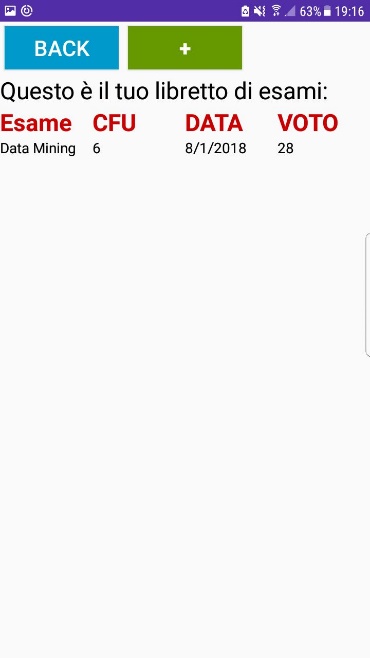
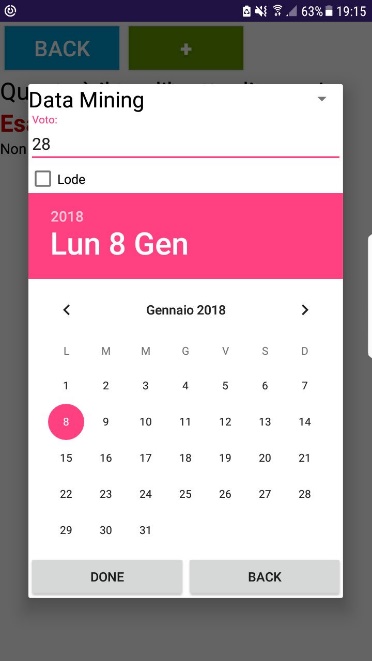
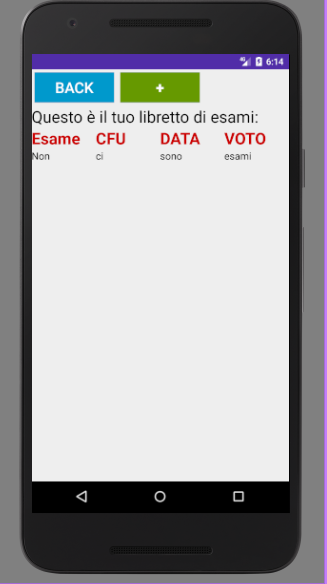
* + **MailProf:** Tale classe permette all’utente di inviare una mail ad un professore. In effetti il compito dell’invio di una mail è demandato ad un client esterno scelto opportunamente dall’utente.



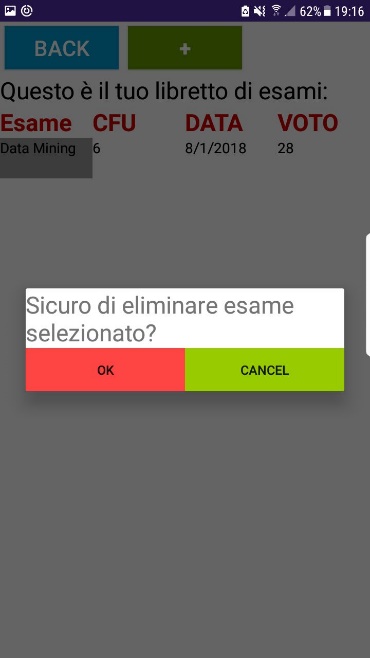
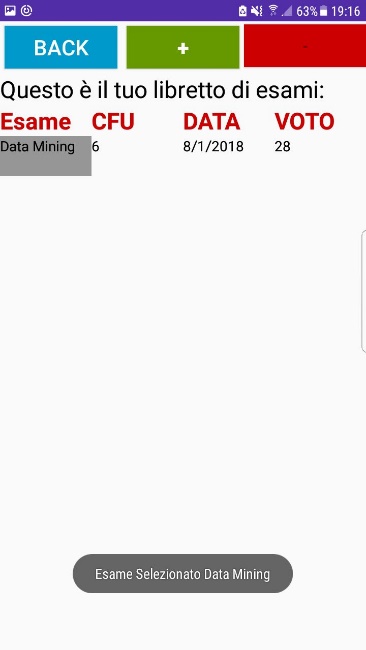
* **LoginActivity:** Tale classe permette ad un utente di effettuare il login. La procedura di login viene effettuata nel metodo attemptLogin() che affida al ControllerLogin il compito di eseguire le operazioni necessarie per l’accesso e per il collegamento con Firebase.



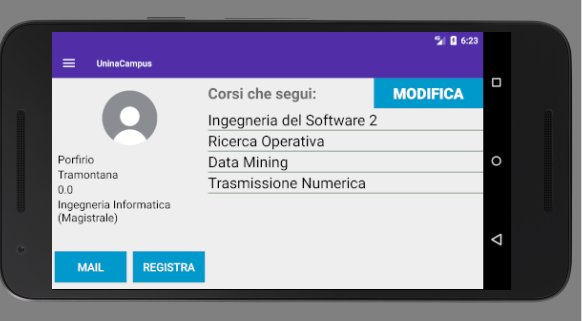
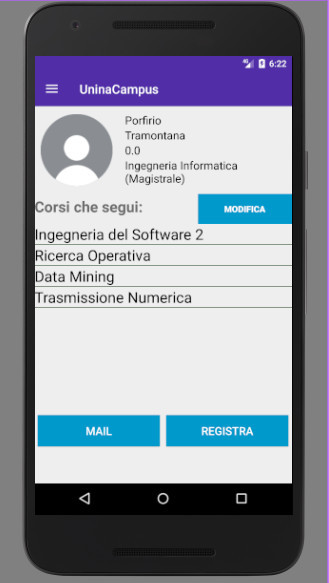
* + **LibrettoActivity:** Tale classe ha il compito di gestire il libretto dell’utente, il quale può aggiungere o eliminare un esame alla sua carriera. Ogni operazione da effettuare sul libretto implica un aggiornamento alla media dello studente. Premendo il bottone “+” è possibile aggiungere un esame.

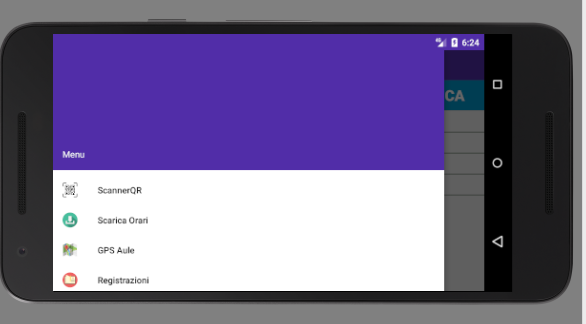
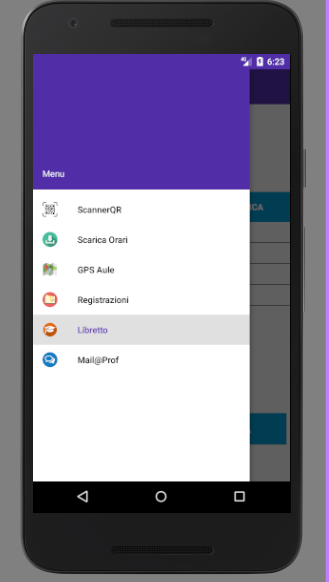


Effettuando un LongClick su un esame è possibile eliminarlo.

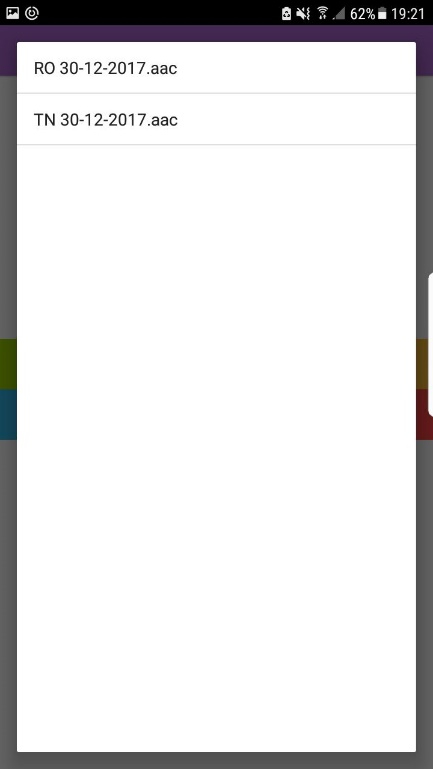


* + **HomePage:** Tale classe costituisce l’home page dell’applicazione, mostrando quindi i dati dell’utente, la lista dei corsi da lui seguiti. È possibile tramite un menu accedere alle funzionalità dell’applicazione stessa e, inoltre, selezionando un corso seguito, è possibile inviare direttamente una mail al professore del corso oppure registrare la lezione.



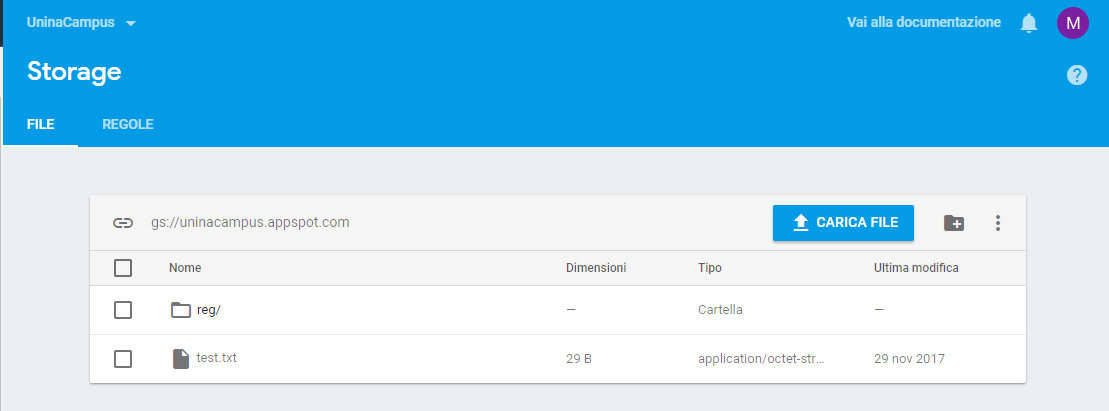


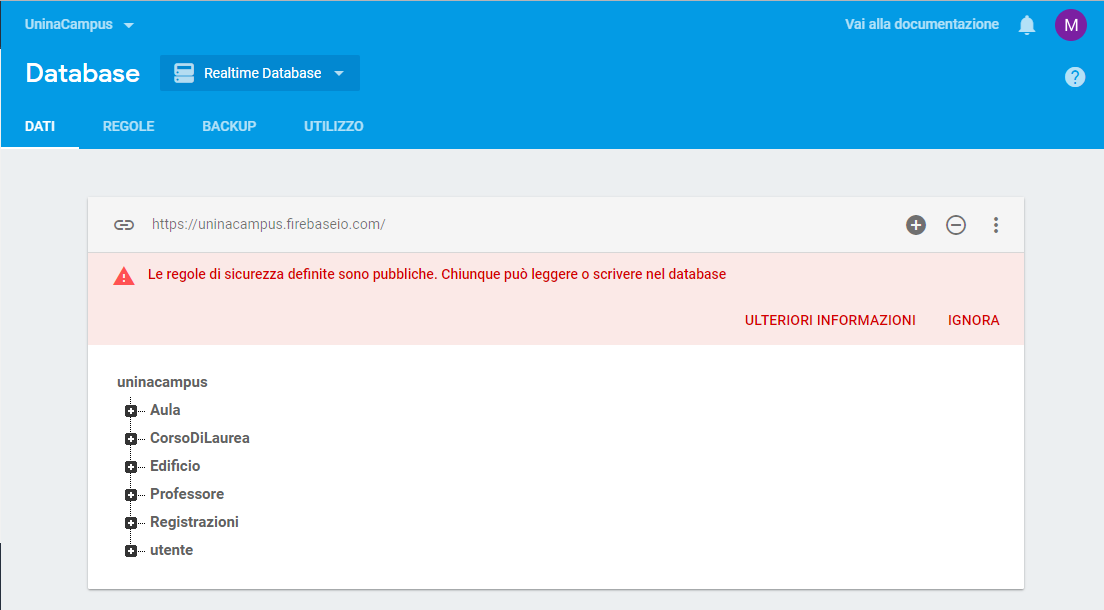
* + **FileChooser:** Tale classe è necessaria per scegliere un file dalla memoria esterna del dispositivo (una registrazione) al fine di effettuare un upload. Le funzionalità di questa classe viene sfruttata dal Recorder Manager.



* ***Package Business***
  + **ControllerAule**: Tale classe è necessaria per gestire le aule dove si svolgono le lezioni.
  + **ControllerCalendario:** Tale classe è necessaria per gestire i permessi per accedere al Calendario del dispositivo e per aggiungere a quest’ultimo gli eventi relativi alle lezioni.
  + **ControllerCorso:** Tale classe ha il compito di gestire i corsi presenti in un corso di laurea.
  + **ControllerLibretto:** Tale classe ha il compito di gestire gli esami dell’utente, in particolare provvede all’inserimento e all’eliminazione di un esame dal libretto prevedendo l’aggiornamento della media e il collegamento con Firebase.
  + **ControllerLogin:** Tale classe ha il compito di gestire il login e il collegamento con Firebase.
  + **ControllerMail:** Tale classe ha il compito di ricavare le email dei professori dei corsi seguiti dell’utente.
  + **ControllerUtente:** Tale classe gestisce le informazioni dell’utente.
  + **Util:** Tale classe è necessaria per contenere funzioni di utilità come la verifica della connessione ad Internet.
* ***Package Entity***
  + **Aula:** rappresenta l’entità “aula”.
  + **Corso:** rappresenta l’entità “corso”.
  + **CorsoDiLaurea:** rappresenta l’entità “corso di laurea”.
  + **Dettagli:** rappresenta l’entità “dettagli” per specificare gli orari dei corsi.
  + **Edificio** rappresenta l’entità “edificio” per specificare la posizione di un’aula.
  + **Esame** rappresenta l’entità “esame”.
  + **Professore** rappresenta l’entità “professore”.
  + **UtenteRegistrato** rappresenta l’entità “utente”.
  1. ***Gestione della persistenza***

La persistenza dei dati è stata gestita con Firebase, un servizio on line, messo a disposizione da Google, che permette di salvare e sincronizzare i dati elaborati da applicazioni web e mobile. Si tratta di un **database NoSQL**dalle grandissime risorse, ad alta disponibilità ed integrabile in tempi molto rapidi. Dopo aver aggiunto un nuovo progetto su Firebase, nella sezione Realtime Database è stata creata una struttura, tramite un file in formato JSON, che rappresenta le entità dell’applicazione. La sezione Storage è, invece, utilizzata come repository per salvare le registrazioni create dall’applicazione.





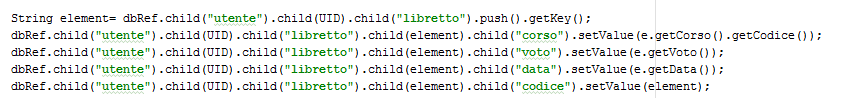
Le funzionalità offerte dall’applicazione necessitano fortemente dei dati memorizzati nel Database e quindi l’interazione con Firebase è indispensabile per il corretto funzionamento. Basta pensare alle procedure di login, di aggiunta di un esame al libretto, oppure di upload di una registrazione o di registrazione, con la quale si crea un nuovo utente.

Esempi di “lettura da Firebase” sono proprio la funzionalità di login, durante la quale vengono letti tutti i dati dell’utente che sta effettuando l’accesso, e la funzionalità di richiesta della lista delle registrazioni online; mentre un esempio di scrittura di nuovi dati è l’aggiunta di un esame nel libretto dello studente.

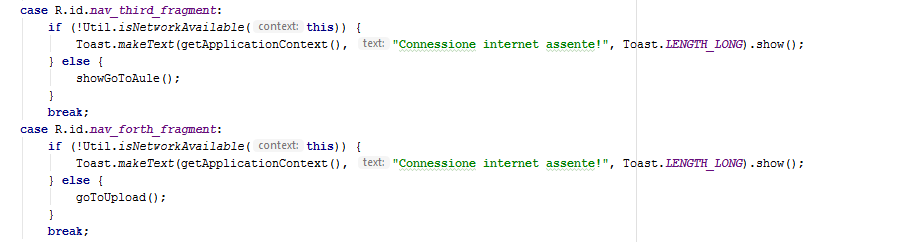
Esempio di lettura (Lista registrazioni)

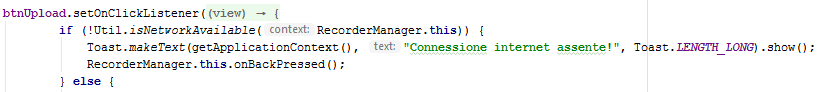


Esempio di Scrittura (Aggiunta esame)



Si è cercato, inoltre, di gestire un’eventuale mancanza di connessione alla rete. In effetti Firebase gestisce autonomamente delle interruzioni di rete. I dati nella cache sono disponibili offline, mentre le scritture sono ritardate quando la rete sarà ripristinata. Tuttavia nella maggior parte dei casi, si è preferito inibire alcune funzionalità nel caso di mancanza di connessione. Ad esempio, partendo dalla HomePage, non è possibile accedere alle funzionalità del Menu, oppure viene inibita la possibilità di effettuare un upload.





1. ***Parte 2: Unit Testing Robolectric.***

***2.1 Introduzione***

Il testing di una applicazione Android può essere effettuato grazie all’utilizzo di strumenti e librerie che consentono una scrittura semplice di test eseguibili che “guidano” direttamente la GUI.

Tra gli strumenti più comuni per la scrittura di test, sono ricordati:

* Android Espresso
* Robotium
* Robolectric

Android Espresso è una libreria sviluppata da Google nel 2014 che permette di scrivere test Junit eseguibili tramite l’esecutore AndroidJUnitRuner. Con Android Espresso è possibile guidare l’esecuzione di eventi direttamente su oggetti della GUI, riferire un oggetto della GUI, eseguire asserzioni su proprietà degli elementi dell’interfaccia. Espresso Test Recorder, gratuito ed integrato con Android Studio, è uno strumento che consente di generare un test per un’applicazione Android con il vantaggio di non scrivere nemmeno una riga di codice, in quanto memorizza le interazioni con un dispositivo e aggiunge asserzioni per verificare gli elementi grafici.

Robotium è un framework a supporto del testing di unità delle activity che estende e potenzia JUnit. In particolare rende più semplice la scrittura di test che riguardano più Activity, Dialog, Toast, Menu, migliorando la leggibilità dei test case. Il Funzionamento di Robotium è tutto basato sull’utilizzo di un oggetto denominato “solo”, tramite cui è possibile interrogare e modificare i widget della UI, eventualmente anche senza conoscerne l’identificativo (ad esempio è possibile selezionare un widget in base al testo che mostra).

I casi di test di sistema o della GUI scritti con Android Espresso sono eseguibili solo su emulatori o dispositivi reali. Una soluzione alternativa che permette di eseguire test senza utilizzare emulatori è rappresentata dal framework Robolectric. In pratica esso mette a disposizione implementazioni alternative (mock) di alcune versioni del framework Android. Scegliendo Robolectric come esecutore di test, questi verranno eseguiti in un ambiente mock, che fa uso di una macchina virtuale locale anziché di emulatori o dispositivi. Il codice viene eseguito su una macchina virtuale Android, che in qualche modo sostituisce automaticamente con mock tutte le classi necessarie all’esecuzione del test.

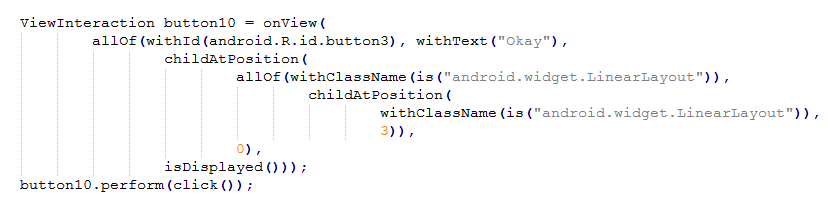
* 1. ***Catalogo di eventi di test base***

I tre framework introdotti precedentemente sono stati usati per scrivere dei test per due applicazioni semplici: MunchLife e TestApplicazione. In effetti considerando i test esistenti per l’applicazione MunchLife e scrivendone dei nuovi sia per MunchLife che per TestApplicazione è stato possibile catalogare gli eventi di test base al variare dei tre strumenti Espresso, Robotium, Robolectric. In particolare TestApplicazione è un progetto Android nato esclusivamente con lo scopo di cogliere quanti più eventi possibili, mentre MunchLife è un’applicazione reale.

Gli eventi catalogati e il modo in cui questi sono gestiti nei tre strumenti sono riportati nel file *MappeNuove.xlsx*. In questo file sono riportati degli esempi di codice di tali eventi ottenuti considerando tutti e tre gli strumenti. Sulla base di quanto riportato in tale file, è stato possibile effettuare una sorta di generalizzazione al fine di creare una corrispondenza tra le funzioni utilizzate da Espresso, Robolectric e Robotium. La generalizzazione è riportata nel file *MappaGenerica.xlsx.* Da come si può notare, Robolectric, a differenza di Espresso e Robotium, ricerca gli elementi tramite il relativo ID e tramite la funzione findViewById() e inoltre, in modo esplicito deve essere specificato se tali elementi si trovano in una activity, dialog o alert dialog. Ciò dipende dal fatto che Espresso e Robotium sono strumenti di testing dinamico e vedono cambiare il contesto di esecuzione, attendendo che il cambiamento sia compiuto prima di cercare di eseguire gli eventi. Inoltre Robotium consente la ricerca di elementi anche avendo a disposizione solo il testo.

Con l’ausilio delle corrispondenze trovate è stato possibile generare dei test in Espresso e Robotium ed effettuare le traduzioni in Robolectric. In particolare per l’applicazione MuchLife è stato creato il test Prova.java tramite il recorder automatico di Espresso e successivamente tale test è stato tradotto al fine di verificare le corrispondenze in Robolectric, creando quindi il test ProvaRobolectric.java. Di seguito sono riportati alcuni esempi di corrispondenze dei due test:

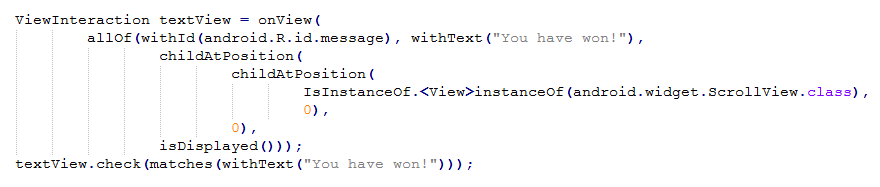
* *Click su bottone Okay Espresso*



*Click su bottone Okay Robolectric*



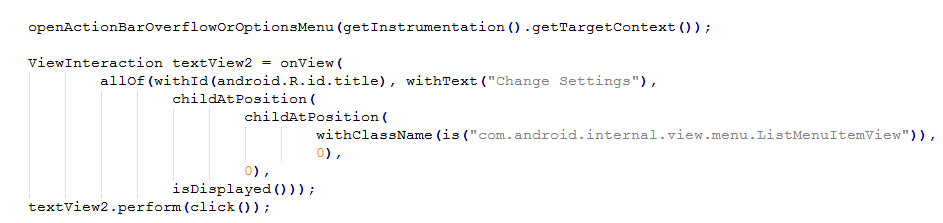
* *Verifica testo Espresso*



*Verifica testo Robolectric*



* *Click su Menu Option Espresso*

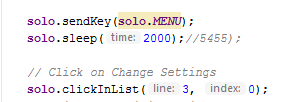


*Click su Menu Option Robolectric*



Per l’applicazione MunchLife è presente un test in Robotium, pertanto è possibile vedere come tale strumento gestisce i vari eventi. Di seguito sono riportate le corrispondenze tra Robotium e Robolectric di alcuni eventi:

* *Click su option Menu Robotium*



*Click su option Menu Robolectric*



* *Click su bottone Robotium*

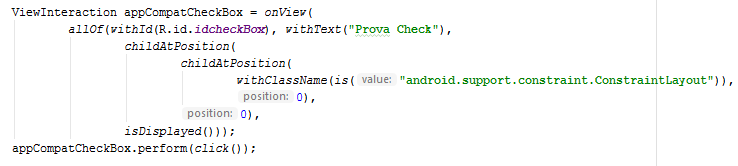


*Click su bottone Robolectric*



Per il progetto TestApplicazione è stato generato tramite il reconder automatico il test in Espresso ProvaTest.java. Successivamente tale test è stato tradotto al fine di utilizzare lo strumento Robolectric, ottenendo quindi ExampleUnitTest.java. Di seguito si vedono le corrispondenze di alcuni eventi:

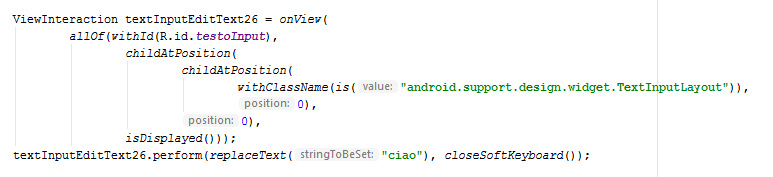
* *Click su CheckBox Espresso*



*Click su CheckBox Robolectric*



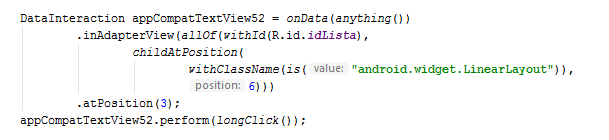
* *Verifica testo Espresso*



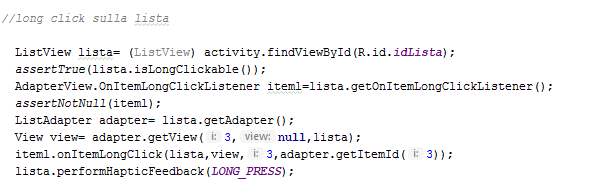
*Verifica Testo Robolectric*



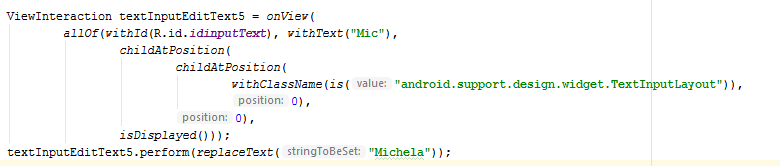
* *LongClick su item di lista Espresso*



*LongClick su item lista Robolectric*



* *Scrittura in un campo di testo Espresso*



*Scrittura di un testo Robolectric*

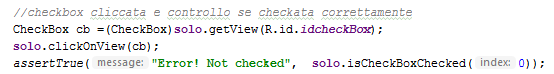


Robotium viene, invece, utilizzato per generare il test specificato nel file RobotiumTest.java, in cui sono riportati gli eventi visti per i test generati per Espresso e Robolectric. Di seguito sono riportati alcuni esempi:

* *Long Click su un elemento di una lista, il corrispondente in Robolectricè analogo a quanto visto prima*



* *Click su checkbox, il corrispondente in Robolectric è analogo a quanto visto prima*



* *Scrittura in un campo di testo, il corrispondente in Robolectric è analogo a quanto visto prima*

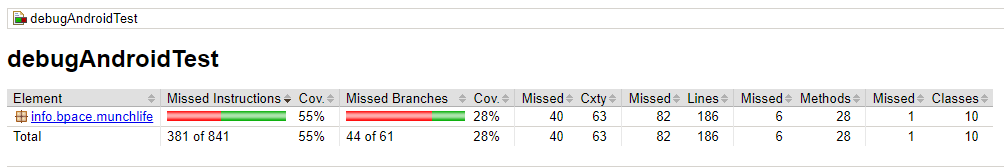
**solo**.enterText((EditText) **solo**.getView(R.id.***testoInput***), **"idinputText!"**);

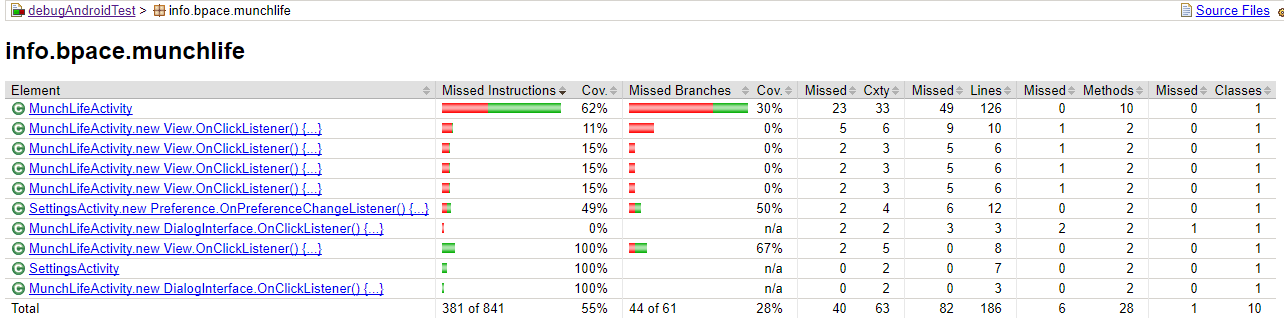
E’ possibile, infine, fare una osservazione riguardante la corrispondenza degli eventi tra i tre strumenti. In effetti, in alcuni casi non è semplice effettuare una generalizzazione in quanto i tre strumenti, pur svolgendo la stessa azione, ricercano l’elemento in maniera diversa. In particolare Robolectric necessita sempre dell’identificativo della risorsa, mentre, Espresso e Robotium possono ricavare un elemento anche dal testo che compare nell’elemento stesso. Un esempio di quanto detto è proprio l’evento di click su un elemento del menu. Quindi, al fine di rendere possibile una traduzione automatica da Espresso/Robotium in Robolectric è necessario inserire informazioni aggiuntive.

* 1. ***Analisi della copertura***

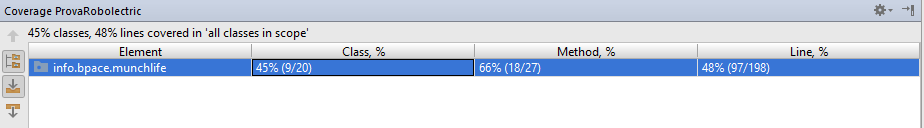
La misura della copertura è ottenuta grazie allo strumento JaCoCo. Si è quindi misurata la copertura dei test scritti sia per l’applicazione MuchLife, che per l’applicazione TestApplicazione.

*Copertura del test Prova.java MunchLife*

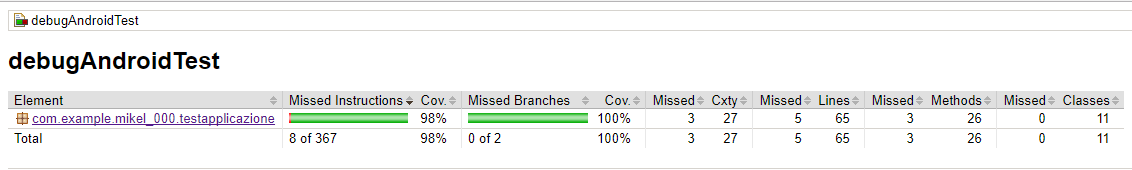




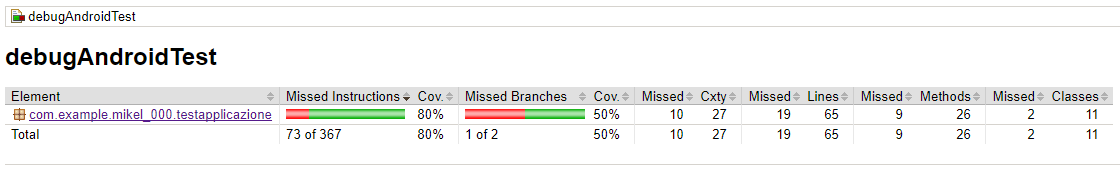
*Copertura del corrispondente test in Robolectric*



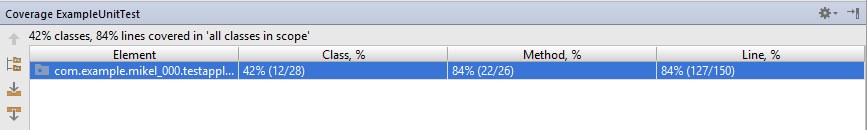
*Copertura del test ProvaTest TestApplicazione*



*Copertura del test RobotiumTest*



*Copertura del corrispondete test in Robolectric*



* 1. ***Progetto Traduttore Automatico***

Dopo aver determinato le corrispondenze dei principali eventi gestiti da Espresso, Robotium e Robolectric, sfruttando quanto riportato in *MappaGenerica.xlsx,* si è cercato di creare un traduttore automatico al fine di trasformare un test ricavato con il recorder automatico di Espresso in un test Robolectric. In effetti a causa della diversità dei due strumenti, non è stato possibile progettare una soluzione che prenda in considerazione tutti gli eventi. In particolare, il traduttore realizzato ha la capacità di gestire e quindi tradurre le seguenti azioni:

* Click su un elemento (bottone, inputText, spinner, CheckBox)
* Long Click su un bottone
* Verifica di corrispondenza di un testo con una stringa
* Verifica che una checkbox sia selezionata
* Verifica dell’esistenza di un elemento
* Inserimento di un testo in un InputText
* Selezione di un elemento dallo spinner
* Selezione di un determinato elemento da una Lista

Una differenza sostanziale tra Espresso e Robolectric consiste nel fatto che il secondo strumento necessita di sapere dove le varie azioni sono dirette, vale a dire su un activity, su un dialog o su un alert dialog. Il traduttore, quindi, prevede la gestione degli eventi indicati precedentemente in tutti e tre i casi di destinazione. Purtroppo, non è possibile determinare l’informazione di destinazione di un evento semplicemente considerando il test in Espresso. Il motivo di ciò dipende dal fatto che Espresso è uno strumento dinamico e pertanto vede cambiare il contesto di esecuzione, attendendo che il cambiamento sia compiuto prima di cercare di eseguire gli eventi. Per risolvere questo problema, la traduzione deve essere guidata dal tester, il quale ha il compito di specificare dove sono diretti i vari eventi. Il traduttore automatico, quindi è formato sostanzialmente da due parti:

1. **Annotator**: uno strumento interattivo che permette al tester di annotare i vari eventi riconosciuti al fine di specificarne la destinazione, che può essere: dialog, activity o alert dialog. In particolare il tester deve specificare //Activity se l’evento deve verificarsi su un activity, altrimenti //Dialog o //AlertDialog
2. **Traduttore**: tale strumento riceve in ingresso un testo annotato e quindi effettua una traduzione in maniera completamente automatica.

In effetti la divisione dei due compiti è stata effettuata al fine di disaccoppiare la fase di annotazione e di traduzione, in modo tale da rendere quest’ultima indipendente dal modo in cui gli eventi sono segnalati.

Al fine di poter utilizzare il traduttore automatico è necessario soddisfare i seguenti prerequisiti:

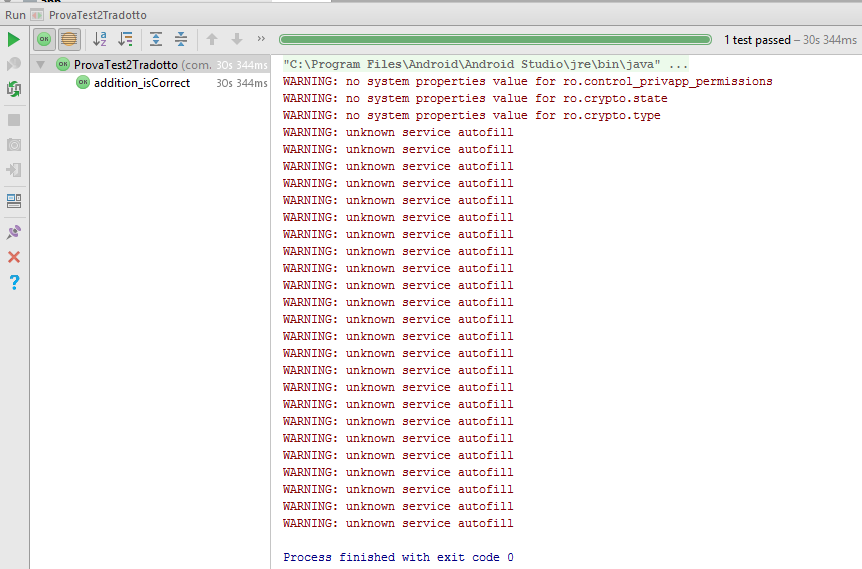
* Il test in Espresso deve essere relativo ad una sola activity, la quale può contenere dialog,e alert dialog
* Nel test in espresso non devono essere presenti le funzioni sleep per gestire la dipendenza temporale degli eventi.
* Il test in Espresso deve prevedere solo gli eventi elencati precedentemente.
* Gli eventi inclusi nel testo devono essere relativi ad oggetti con un identificativo esplicito.
* Il test in Espresso deve contenere solo una funzione di test, ovvero solo una funzione con annotazione @Test.

Se tali prerequisiti sono rispettati è possibile effettuare una traduzione automatica seguendo i seguenti passi.

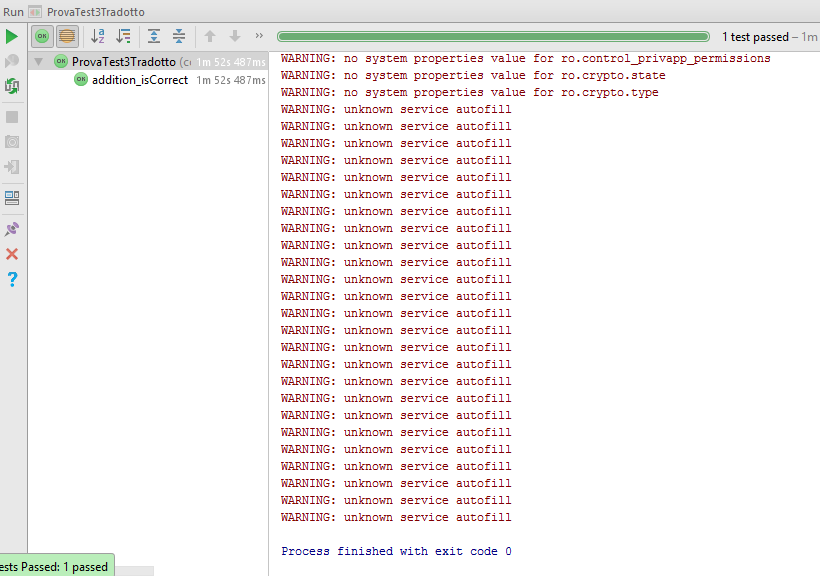
1. Generare un test in Espresso, rispettando i prerequisiti visti.
2. Eseguire a linea di comando la classe Main.java del progetto Annotator specificando in ingresso il path del file da annotare. Il programma genera un file nella stessa cartella del file di input con il nome FilediInput\_annotato.java. A questo punto è possibile porre in ingresso al programma Traduttore il file ottenuto.
3. Per fare ciò è necessario aprire il progetto in Eclipse, modificare la variabile file che identifica il file di output.
4. Modificare la variabile allLines specificando il path del file di input.
5. Modificare la variabile allLines2 specificando il percorso del file RobolectricTest.java, necessario in quanto costituisce la struttura di un file di Test Robolectric.
6. Avviare l’esecuzione.
7. Rinominare il file e la classe in esso contenuta in modo che i nomi coincidano.
8. Importare il file di output nel progetto Android.
9. Verificare il funzionamento, risolvendo eventuali problemi di import.

**Esempi di Funzionamento**

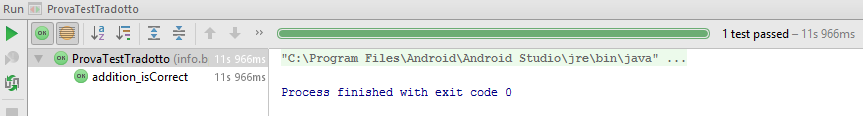
* **Esempio per TestApplicazione**: Tramite il recorder automatico di Espresso è stato generato il test ProvaTest2.java. Seguendo la procedura descritta precedentemente, viene generato il test in Robolectric ProvaTest2Tradotto.java



* **Secondo esempio per TestApplicazione:** Tramite il recorder automatico di Espresso è stato generato il test ProvaTest3.java. Seguendo la procedura descritta precedentemente, viene generato il test in Robolectric ProvaTest3Tradotto.java



* **EsempioMunchlife**: Tramite il recorder automatico di Espresso è stato generato il test ProvaTest.java. Seguendo la procedura descritta precedentemente, viene generato il test in Robolectric ProvaTestTradotto.java



* **Esempio Trolly**: Tramite il recorder automatico di Espresso è stato generato il test TrollyTest.java. Seguendo la procedura descritta precedentemente, viene generato il test in Robolectric TestRobolectric.java. La traduzione viene portata a termine correttamente, ma è stato riscontrato un problema con l’oggetto lista, essendo i suoi elementi generati in maniera dinamica. A quanto pare, Robolectric non riesce ad accedervi, restituendo sempre una lista vuota.

