UNIVERSITÀ DEGLI STUDI DI SALERNO

Corso di Laurea in Informatica, a.a. 2022-23
Progetto del corso di Ingegneria del Software
prof. A. De Lucia, prof. M. De Stefano
Repository GitHub: https://github.com/giocolella/is-bookbearer-22-23

Book Bearer

- Test Plan Document | Versione 1.0

Coordinatore del progetto:

Nome	**Matricola**
De Lucia Andrea	
De Stefano Manuel	
Colella Giorgio	0512105946

Partecipanti:

Nome	**Matricola**
Colella Giorgio	0512105946

Scritto da:	Colella Giorgio

Revision History

Data	Versione	Descrizione	Autore

Indice

1.Introduzione

Il test plan è un documento che si concentra sull'aspetto manageriale del testing e nasce per la definizione e pianificazione dell'attività di testing dell'applicazione. L'idea è quella di individuare anomalie o fault nel software che possono causare un comportamento errato e non conforme alle specifiche fornite nei documenti precedenti. Qui si definirà cosa verrà e cosa non verrà testato, i criteri che si usano per considerare un test fallito o passato, i requisiti hardware/software per effettuare i test,ecc…

2.Relazioni con altri documenti

Fare riferimento al RAD per la specifica delle funzionalità, al SDD per la definizione dei sottosistemi e all'ODD per la definizione delle operazioni.

3.Overview del sistema

Il sistema è strutturato in package che rappresentano le funzionalità dei diversi sottosistemi. Tali package seguono la struttura MVP che permette di favorire il disaccoppiamento. Gli elementi di un pacchetto sono indipendenti da quelli di un altro.

4.Funzionalità da testare e da non testare

Ciò che verrà testato sono i requisiti funzionali:

- Gestione Registrazione;

- Gestione Autenticazione;

- Gestione Area Utente;

- Gestione Liste;

- Gestione Recensioni;

- Gestione Catalogo.

Ciò che non verrà testato sono i requisiti non funzionali.

5.Criteri pass/fail del test

Le definizioni di test fallito e test passato sono in continuo cambiamento. In questo caso, essendo il senso dell'esistenza di un test quello di individuare delle fault, un test viene considerato passato se individua una fault. Nel caso in cui quest'ultima venisse individuata verrà prima controllato il test per assicurarci che sia scritto bene e, in caso lo sia, si controllerà il codice di produzione per effettuare le dovute modifiche le quali verranno documentate. Poi verrà fatto un test di regressione per verificare che tale cambiamento non abbia indotto errori in altre componenti già testate.

6.Approccio

Il testing di Android ha alcune differenze rispetto ad un tipico test di prodotti di web development (questo punto verrà approfondito nel Test Execution Report) :

1. È possibile effettuare lo unit testing della View (Activity);
2. Il testing di integrazione ed End-to-End vengono messi da Google nello stesso "pacchetto" chiamato Instrumented tests. In questo caso entrambi i tipi di testing necessitano di un device emulato per essere eseguiti e quindi a volte sono complementari ed altre volte mutualmente esclusivi (portando in questo caso ad una diminuzione dei costi/tempi di testing).
   Esempio: Nel caso della registrazione di un utente il testing E2E viene utilizzato per verificare che tutte le view (TextView, EditText, Toast, ecc…) siano inizializzate/mostrate nel modo corretto e che il flusso di esecuzione delle diverse Activity sia quello corretto.
   Tuttavia le view non ci permettono di sapere cosa è successo nel database. Il test di integrazione mostrerà ancora una volta l'esecuzione di una registrazione nel device emulato senza però curarsi delle View e verificando che l'utente sia stato effettivamente registrato nel database e con i dati corretti.
   Nel caso della ricerca di un libro e della visualizzazione delle informazioni relative al libro selezionato, le asserzioni fatte alle diverse View nel testing E2E sono sufficienti a verificare che i dati ottenuti dal database sono corretti. Un testing di integrazione in questo caso verrebbe comunque eseguito su device emulato senza fornire alcuna nuova asserzione e diventando quindi ridondante.

3. Il testing è strutturato in due package forniti da Android studio: il package indicato con “(test)” dove non viene utilizzato il device emulato e si possono effettuare i test di unità ed il package indicato con “(androidTest)” dove si ottiene il vero contesto dell’applicazione e si utilizza un device emulato per effettuare test di integrazione/E2E.

Le fasi di testing saranno quindi:

1. **Unit testing** dei validatori e della View;
2. **Instrumented test** per testare il funzionamento delle diverse componenti quando interagiscono tra di loro (integration) e che le View vengano inizializzate/mostrate correttamente nel flusso di esecuzione di un’interazione utente simulata (End-to-End).

L’approccio all’instrumented testing sarà purtroppo top-down e cioè si partirà dal layer più alto. Ogni sottosistema del layer più alto viene testato individualmente e poi quelli del layer successivo vengono testati chiamando i sottosistemi precedenti e così via fino a testare tutti i sottosistemi. Questo perché gli instrumented tests utilizzano un emulatore e partono sempre da una View (layer di presentazione). Questo rende estremamente difficile creare dei Test Stub. Se poi si considera l’asincronicità di Firebase, la complessità aumenta. Non è stato possibile utilizzare un’approccio sandwich perché Firebase necessita della View effettiva e quindi testare il Model col Presenter non è fattibile perché manca la View.

I casi di test verranno selezionati utilizzando “Category Partition” che permette di decomporre lo spazio di input in categorie che vengono poi partizionate in classi di equivalenza per ottenere un testing efficiente e privo di ridondanze.

7.Sospensione e ripresa

La fase di test viene sospesa appena si incontra una fault e può essere ripresa appena risolto il problema incontrato e appena si ci è assicurati che la consistenza dei dati nel database è mantenuta.

8.Materiale per il testing

Gli strumenti utilizzati sono:

1. Firebase Firestore, Firebase Authentication e Firebase Storage;
2. Android device emulato Pixel XL con Android 9.0;
3. Robolectric per lo unit testing della View;
4. Mockito;
5. JUnit4 per il testing di integrazione ed End-to-End perché Google non ha ancora implementato il supporto a JUnit5 **negli instrumented tests** (https://issuetracker.google.com/issues/127100532) e le librerie di terze parti sono improntate a Kotlin;
6. JUnit5 per lo unit testing dei validatori;
7. Gradle;
8. Espresso Test Recorder per gli instrumented tests;
9. Android Test Orchestrator per risolvere alcune problematiche di Espresso Test Recorder.

9.Casi di test

-Unit testing dei validatori (+ Instrumented testing per quando si accede al database)

Gestione Registrazione (TC_Registrazione)

Parametro: nomeUtente	
Categorie	Scelte
Lunghezza nomeUtente - LN	1.Lunghezza = 0 – campo vuoto [errore] 2.Lunghezza > 0 – [property **validLNValue**]

Parametro: email Formato email: deve contenere “@”	
Categorie	Scelte
Lunghezza email - LE	1.Lunghezza = 0 – campo [errore] 2.Lunghezza > 0 – [property **validLEValue**]
Formato email - FE	1.Formato incorretto [if validLEValue] [errore] 2.Formato corretto [if validLEValue] [property **validFEValue**]
Email già presente - EGP	1.Email già presente [if validFEValue] [errore] 2.Email non presente [if validFEValue] [property **ENPValue**]

Parametro: password Formato: almeno 8 caratteri (l’unica limitazione di Firebase Authentication riguarda la lunghezza)	
Categorie	Scelte
Lunghezza password - LP	1.Lunghezza = 0 – campo [errore] 2.Lunghezza > 0 – [property **validLPValue**]
Formato password - FP	1.Formato incorretto [if validLPValue] [errore] 2.Formato corretto [if validLPValue] [property **validFPValue**]

Parametro: confermaPassword	
Categorie	Scelte
Lunghezza confermaPassword - LCP	1.Lunghezza = 0 – campo vuoto [errore] 2.Lunghezza > 0 – [property **validLCPValue**]
Equivalenza con password – ECP	1.Equivalenza assente [if validLCPValue] [errore] 2.Equivalenza presente [if validLCPValue] [property **EQ**]

Codice	Combinazione	Esito
TC_0.1INT	LN2-LE2-FE2-EGP2-LP2-FP2-LCP2-ECP2	“Registrazione completata”
TC_0.2	LN1-LE1-FE2-EGP2-LP1-FP2-LCP1-ECP2	“Potrebbero esserci dei campi vuoti”
TC_0.3	LN1-LE2-FE2-EGP2-LP2-FP2-LCP2-ECP2	“Potrebbero esserci dei campi vuoti”
TC_0.4	LN2-LE1-FE2-EGP2-LP2-FP2-LCP2-ECP2	“Potrebbero esserci dei campi vuoti”
TC_0.5	LN2-LE2-FE2-EGP2-LP1-FP2-LCP2-ECP2	“Potrebbero esserci dei campi vuoti”
TC_0.6	LN2-LE2-FE2-EGP2-LP2-FP2-LCP1-ECP2	“Potrebbero esserci dei campi vuoti”
TC_0.7E2E	LN2-LE2-FE2-EGP1-LP2-FP2-LCP2-ECP2	“Email già esistente”
TC_0.8	LN2-LE2-FE2-EGP2-LP2-FP1-LCP2-ECP2	“Password troppo corta”
TC_0.91	LN2-LE2-FE1-EGP2-LP2-FP2-LCP2-ECP2	“Email incorretta”
TC_0.92	LN2-LE2-FE2-EGP2-LP2-FP2-LCP2-ECP1	“Password e conferma password non coindicono”

Gestione Autenticazione (TC_Login)

Parametro: email	
Categorie	Scelte
Lunghezza email - LEL	1.Lunghezza = 0 – campo vuoto [errore] 2.Lunghezza > 0 – [property **validLELValue**]
Email presente - EP	1.Email non presente [if validLELValue] [property **ENPLValue**] 2.Email presente [if validLELValue] [property **EPValue**]

Parametro: password	
Categorie	Scelte
Lunghezza password - LPL	1.Lunghezza = 0 – campo vuoto [errore] 2.Lunghezza > 0 – [property **validLPLValue**]
Password corretta - PC	1.Password incorretta [if EPValue] [if validLPLValue] [errore] 2.Password corretta [if EPValue] [if validLPLValue] [property **PCValue**]

Codice	Combinazione	Esito
TC_1.0INT	LEL2-EP2-LPL2-PC2	“Login effettuato”
TC_1.2	LEL1-EP2-LPL2-PC2	“Potrebbero esserci dei campi vuoti”
TC_1.3	LEL2-EP2-LPL1-PC2	“Potrebbero esserci dei campi vuoti”
TC_1.4	LEL1-EP2-LPL1-PC2	“Potrebbero esserci dei campi vuoti”
TC_1.5E2E	LEL2-EP1-LPL2-PC2	“Credenziali errate”

TC_1.6E2E	LEL2-EP2-LPL2-PC1	“Credenziali errate”

Gestione Area Utente (TC_AreaUtente)

TC_ModificaMail

Parametro: newMail Formato email: deve contenere “@”	
Categorie	Scelte
Lunghezza email - LEM	1.Lunghezza = 0 – campo vuoto [errore] 2.Lunghezza > 0 – [property **validLEMValue**]
Formato email - FEM	1.Formato incorretto [if validLEMValue] [errore] 2.Formato corretto [if validLEMValue] [property **validFEMValue**]
Email già presente - EGPM	1.Email già presente [if validFEMValue] [errore] 2.Email non presente [if validFEMValue] [property **ENPMValue**]

Parametro: password Formato: almeno 8 caratteri (l’unica limitazione di Firebase Authentication riguarda la lunghezza)	
Categorie	Scelte
Lunghezza password - LPMM	1.Lunghezza = 0 – campo vuoto [errore] 2.Lunghezza > 0 – [property **validLPMMValue**]
Formato password - FPMM	1.Formato incorretto [if validLPMMValue] [errore]

	2.Formato corretto [if validLPMMValue] [property **validFPMMValue**]
Password corretta - PCMM	1.Password incorretta [if validFPMMValue] [errore] 2.Password corretta [if validFPMMValue] [property **PCValue**]

Codice	Combinazione	Esito
TC_2.1INT	LEM2-FEM2-LPMM2-PCMM2-FPMM2-EGPM2	“Email modificata”
TC_2.2	LEM1-FEM2-LPMM1-PCMM2-FPMM2-EGPM2	“Potrebbero esserci dei campi vuoti”
TC_2.3	LEM1-FEM2-LPMM2-PCMM2-FPMM2-EGPM2	“Potrebbero esserci dei campi vuoti”
TC_2.4	LEM2-FEM2-LPMM1-PCMM2-FPMM2-EGPM2	“Potrebbero esserci dei campi vuoti”
TC_2.5	LEM2-FEM1-LPMM2-PCMM2-FPMM2-EGPM2	“Email incorretta”
TC_2.6E2E	LEM2-FEM2-LPMM2-PCMM2-FPMM2-EGPM1	“Email già esistente”
TC_2.7	LEM2-FEM2-LPMM2-PCMM2-FPMM1-EGPM2	“Password troppo corta”
TC_2.8E2E	LEM2-FEM2-LPMM2-PCMM1-FPMM2-EGPM2	“Password sbagliata”

TC_ModificaPassword

Parametro: newPassword Formato: almeno 8 caratteri (l'unica limitazione di Firebase Authentication riguarda la lunghezza)	
Categorie	Scelte
Lunghezza password – LPM	1.Lunghezza = 0 – campo vuoto [errore] 2.Lunghezza > 0 – [property **validLPMValue**]
Formato password - FPM	1.Formato incorretto [if validLPMValue] [errore] 2.Formato corretto [if validLPMValue] [property **validFPMValue**]

Parametro: oldPassword Formato: almeno 8 caratteri (l'unica limitazione di Firebase Authentication riguarda la lunghezza)	
Categorie	Scelte
Lunghezza password – LOP	1.Lunghezza = 0 – campo vuoto [errore] 2.Lunghezza > 0 – [property **validLOPValue**]
Formato password – FOP	1.Formato incorretto [if validLOPValue] [errore] 2.Formato corretto [if validLOPValue] [property **validFOPValue**]
Password corretta - POC	1.Password incorretta [if validFOPValue] [errore] 2.Password corretta [if validFOPValue] [property **POCValue**]

Codice	Combinazione	Esito
TC_3.1E2E	LPM2-FPM2-LOP2-FOP2-POC2	“Password modificata”
TC_3.2	LPM1-FPM2-LOP1-FOP2-POC2	“Potrebbero esserci dei campi vuoti”
TC_3.3	LPM1-FPM2-LOP2-FOP2-POC2	“Potrebbero esserci dei campi vuoti”
TC_3.4	LPM2-FPM2-LOP1-FOP2-POC2	“Potrebbero esserci dei campi vuoti”
TC_3.5	LPM2-FPM1-LOP2-FOP2-POC2	“Nuova Password troppo corta”
TC_3.6	LPM2-FPM2-LOP2-FOP1-POC2	“Vecchia Password troppo corta”
TC_3.7E2E	LPM2-FPM2-LOP2-FOP2-POC1	“Vecchia password sbagliata”

Gestione Recensioni (TC_Recensioni)

TC_RecensioneAggiunta

Il punteggio è selezionato in un RadioButton con uno switch che fornisce numeri da 1 a 5 (con valore di default “NP” e cioè Non Pervenuto).

Parametro: punteggio	
Categorie	Scelte
Lunghezza punteggio – LPR	1.Lunghezza = 2 – NP [errore] 2.Lunghezza = 1 – [property **validLPRValue**]

Parametro: descrizione	
Categorie	Scelte
Lunghezza descrizione – LD	1.Lunghezza = 0 – campo vuoto [errore] 2.Lunghezza > 0 – [property **validLDValue**]

L'id Utente è creato e gestito da Firebase Authentication ed usato per verificare (come indicato nell'ODD) che la recensione che l'utente sta cercando di aggiungere ad un determinato libro sia la prima.

Parametro: idUtente	
Categorie	Scelte
Id Utente già presente – IUP	1.Id Utente già presente [if validLPRValue] [if validLDValue] [errore] 2.Id Utente non presente [if validLPRValue] [if validLDValue] [property **IUNPValue**]

Codice	Combinazione	Esito
TC_4.1INT	LPR2-LD2-IUP2	"Recensione aggiunta"
TC_4.2	LPR1-LD1-IUP2	"Potrebbero esserci dei campi vuoti"
TC_4.3	LPR1-LD2-IUP2	"Potrebbero esserci dei campi vuoti"
TC_4.4	LPR2-LD1-IUP2	"Potrebbero esserci dei campi vuoti"
TC_4.5E2E	LPR2-LD2-IUP1	"Recensione già aggiunta"

Gestione Liste (TC_Liste)

TC_ListeAggiuntaAListaDaLeggere

Quando un utente vuole aggiungere un libro alla lista dei libri da leggere, l'ISBN di tale libro viene preso dal database (quindi grazie a TC_CatalogoAggiuntaLibro non dobbiamo verificare la sua lunghezza ed il suo formato) ed usato per vedere se è presente nel documento ListaDaLeggere o in quello ListaLetti dell'utente. Se è presente in uno dei due documenti il libro non viene aggiunto al documento ListaDaLeggere.

Parametro: ISBN	
Categorie	Scelte
ISBN già presente - INGP	1.ISBN già presente in ListaDaLeggere [errore] 2.ISBN già presente in ListaLetti [errore] 3.ISBN non presente [property **INGPValue**]

Codice	Combinazione	Esito
TC_5.1E2E	INGP3	“Libro aggiunto a lista da leggere”
TC_5.2E2E	INGP1	“Libro già presente in lista da leggere”
TC_5.3E2E	INGP2	“Libro già presente in lista letti”

Gestione Catalogo (TC_Catalogo)

TC_CatalogoAggiuntaLibro

Parametro: ISBN Formato ISBN: deve essere di esattamente 14 caratteri	
Categorie	Scelte
Lunghezza ISBN - LI	1.Lunghezza = 0 – campo vuoto [errore] 2.Lunghezza > 0 – [property **validLIValue**]
Formato ISBN - FI	1.Formato incorretto [if validLIValue] [errore] 2.Formato corretto [if validLIValue] [property **validFIValue**]
ISBN già presente - ISGP	1.ISBN già presente [if validFIValue] [errore] 2.ISBN non presente [if validFIValue] [property **ISGPValue**]

Parametro: titolo	
Categorie	Scelte
Lunghezza titolo - LT	1.Lunghezza = 0 – campo vuoto [errore] 2.Lunghezza > 0 – [property **validLTValue**]

Parametro: autore	
Categorie	Scelte
Lunghezza autore - LA	1.Lunghezza = 0 – campo vuoto [errore] 2.Lunghezza > 0 – [property **validLAValue**]

Parametro: genere	
Categorie	Scelte
Lunghezza genere - LG	1.Lunghezza = 0 – campo vuoto [errore] 2.Lunghezza > 0 – [property **validLGValue**]

Parametro: bookImg	
Categorie	Scelte
Lunghezza bookImg - LB	1.Lunghezza = 0 – campo vuoto [errore] 2.Lunghezza > 0 – [property **validLBValue**]

Parametro: annoUscita Formato annoUscita: “^[0-9]*$” con almeno 4 cifre	
Categorie	Scelte
Lunghezza annoUscita - LAU	1.Lunghezza = 0 – campo vuoto [errore] 2.Lunghezza > 0 – [property **validLAUValue**]
Formato annoUscita - FAU	1.Formato incorretto [if validLAUValue] [errore] 2.Formato corretto [if validLAUValue] [property **validFAUValue**]

Parametro: NumPagine Formato NumPagine: “^[0-9]*$”	
Categorie	Scelte
Lunghezza NumPagine – LNP	1.Lunghezza = 0 – campo vuoto [errore] 2.Lunghezza > 0 – [property **validLNPValue**]
Formato NumPagine – FNP	1.Formato incorretto [if validLNPValue] [errore] 2.Formato corretto [if validLNPValue] [property **validFNPValue**]

Codice	Combinazione	Esito
TC_6.1INT	LI2-FI2-ISGP2-LT2-LA2-LG2-LB2-LAU2-FAU2-LNP2-FN2	“Libro aggiunto al catalogo”
TC_6.2	LI1-FI2-ISGP2-LT1-LA1-LG1-LB1-LAU1-FAU2-LNP1-FN2	“Potrebbero esserci dei campi vuoti”

TC_6.3	LI1-FI2-ISGP2-LT2-LA2-LG2-LB2-LAU2-FAU2-LNP2-FN2	“Potrebbero esserci dei campi vuoti”
TC_6.4	LI2-FI2-ISGP2-LT1-LA2-LG2-LB2-LAU2-FAU2-LNP2-FN2	“Potrebbero esserci dei campi vuoti”
TC_6.5	LI2-FI2-ISGP2-LT2-LA1-LG2-LB2-LAU2-FAU2-LNP2-FN2	“Potrebbero esserci dei campi vuoti”
TC_6.6	LI2-FI2-ISGP2-LT2-LA2-LG1-LB2-LAU2-FAU2-LNP2-FN2	“Potrebbero esserci dei campi vuoti”
TC_6.7	LI2-FI2-ISGP2-LT2-LA2-LG2-LB2-LAU1-FAU2-LNP2-FN2	“Potrebbero esserci dei campi vuoti”
TC_6.8	LI2-FI2-ISGP2-LT2-LA2-LG2-LB1-LAU2-FAU2-LNP2-FN2	“Potrebbero esserci dei campi vuoti”
TC_6.91	LI2-FI2-ISGP2-LT2-LA2-LG2-LB2-LAU2-FAU2-LNP1-FN2	“Potrebbero esserci dei campi vuoti”
TC_6.92E2E	LI2-FI2-ISGP1-LT2-LA2-LG2-LAU2-FAU2-LNP2-FN2	“Libro già aggiunto”
TC_6.93	LI2-FI2-ISGP2-LT2-LA2-LG2-LAU2-FAU1-LNP2-FN2	“Anno d’uscita incorretto”
TC_6.94	LI2-FI2-ISGP2-LT2-LA2-LG2-LAU2-FAU2-LNP2-FN1	“Numero pagine incorretto”

-Unit testing della view

In questo caso si testerà:

1. L'effettiva inizializzazione delle View (EditText, TextView, ImageView, ecc…);
2. I metodi che instradano un Activity verso un'altra Activity assicurandoci che quella nuova sia effettivamente quella che ci aspettiamo e, nel caso sia richiesto, che quella precedente venga tolta dallo stack delle View;
3. I metodi che mostrano i Toast;
4. I metodi che popolano i Custom Adapter delle ListView.

-Integration testing

Considerando il package "com.example.bookbearer.addreview" abbiamo: la classe AddReviewActivity (la View), la classe AddReviewPresenter e la classe AddReviewModel.

A causa della forte dipendenza di Android con la View, l'idea è quella di partire dall'interazione tra la View ed il Presenter creando uno Stub o Repository per il Model. Successivamente si elimina lo Stub e si verifica l'effettiva interazione di tutte e tre queste componenti.

I diversi package sono indipendenti tra di loro quindi questo approccio lo si può applicare a tutti i rimanenti.

-End-to-End testing

Per il testing E2E si fa riferimento ai diversi casi d'uso indicati nel RAD. Questi saranno dei punti guida nella simulazione dell'interazione di un utente simulato.