UNIVERSITA' DEGLI STUDI DI NAPOLI FEDERICO II
FACOLTÀ DI INFORMATICA

Ingegneria del Software 2022/2023

Ratatouille23

Docenti:
Prof. Sergio Di Martino
Prof. Francesco Cutugno
Prof. Luigi Lucio Libero Starace

Gruppo INGSW2223_N_04

No.	Nome e Cognome	Matricola
1	Luigi Vessella	N86003354
2	Matteo Marino	N86003963
3	Biagio Speranza	N86002964

Indice

1 Descrizione del progetto **4**

2 Testing JUnit **5**

2.1 Test funzione check credenziali 5
2.2 Test funzione che calcola incasso in un range di date. 8
2.3 Test funzione che controlla i campi di un ristorante. 13
2.4 Testing delle funzione media (in statistiche). 16
2.5 Usabilità sul campo 19
2.5.1 Premessa 19
2.5.2 Presentazione degli utenti 19
2.5.3 Il confronto con gli utenti 19
2.5.4 Valutazione finale 20

1 Descrizione del progetto

Ratatouille23 è un sistema finalizzato al supporto alla gestione e all'operatività di attività di ristorazione. Il sistema consiste in un'applicazione performante e affidabile, attraverso cui gli utenti possono fruire delle funzionalità del sistema in modo intuitivo, rapido e piacevole. La nostra visione della richiesta prevede lo sviluppo di un'applicazione mobile su sistema operativo Android che offrirà agli utilizzatori i seguenti servizi:

- *I proprietari (o amministratori) potranno creare account per i dipendenti*
- *I proprietari saranno in grado di gestire uno o più ristoranti*
- *Si avrà la possibilità di visualizzare e/o modificare il menu*
- *I dipendenti (camerieri) saranno in grado di prendere e inoltrare le ordinazioni in cucina*
- *Gli addetti alla cucina potranno avvisare i camerieri nel momento in cui è pronta un'ordinazione*
- *Tutti potranno visionare lo storico delle ordinazioni con i dettagli*

Ovviamente, il tipo di funzionalità messo a disposizione dall'applicativo sarà cambiato dinamicamente a seconda di chi si logga. Gli amministratori e/o supervisori saranno inoltre dotati di tablet con a bordo l'OS di Google Android per una migliore fruizione della loro dashboard.

I dipendenti quali camerieri, operatori di cucina, capisala, saranno dotati di smartphone aziendali sempre con OS Android correttamente configurati per un'ottimale fruizione dell'applicazione.

Tutti i dispositivi dovranno essere in grado di accedere a internet, preferibilmente per tutta la durata del servizio. Il funzionamento del server è invece garantito da servizi allo stato dell'arte quali Microsoft Azure.

2 Testing JUnit

In questa sezione, presentiamo una delle fasi cruciali all'interno del ciclo di sviluppo del software, ovvero il testing. Il testing può assumere diverse forme e tipologie, ma per la semplicità della presentazione, in questa sede si richiede il testing di soli 4 metodi con due o più parametri, che verrà effettuato mediante l'utilizzo del framework JUnit 5.

2.1 Test funzione check credenziali

Il metodo *checkCredentials* ha il compito di verificare la validità di un indirizzo email e di una password. In particolare, il metodo accetta due stringhe come input: *mail*, che rappresenta l'indirizzo email, e *pswrd*, che rappresenta la password. Il metodo restituisce un oggetto *ArrayList* di interi, che rappresenta una lista di eventuali errori verificatisi durante l'esecuzione del metodo. In altre parole, il metodo controlla se l'indirizzo email e la password rispettano determinati requisiti, e in caso contrario, aggiunge il corrispondente codice di errore alla lista. Infine, il metodo restituisce la lista degli errori, eventualmente vuota se non sono stati riscontrati problemi.

```
1
2       public class CheckCredentialsTest {
3
4       /*
5       CLASSI DI EQUIVALENZA:
6
7       PASSWORD: {VALIDA, VUOTA, TROPPO CORTA, NON RISPETTA LA REGEX}
8         - VALIDA = Almeno 8 caratteri di cui 1 Maiuscola,
9         1 Minuscola, 1 Carattere speciale, 1 numero
10
11      EMAIL: {VALIDA, VUOTA, NON VALIDA}
12        - VALIDA = Deve rispettare la sintassi
13        di un email corretta
14
15  ------------------------------------------------------------------------------
16
17      CODICI DI ERRORE:
18        9 = PASSWORD VUOTA,
19        10 = PASSWORD TROPPO CORTA,
20        11 = PASSWORD NON VALIDA (REGEX)
21        12 = EMAIL VUOTA,
22        13 = EMAIL NON VALIDA
23
24  ------------------------------------------------------------------------------
25
26      STRATEGIE DI TESTING UTILIZZATE:
27        BlackBox secondo il criterio SECT
28
29      CASI DI TESTING INDIVIDUATI:
30        22 in 12 metodi
31
```

```
32  ------------------------------------------------------------------------------ */
33
34      public ArrayList<Integer> codici_errore = new ArrayList<Integer>();
35
36      @AfterEach
37      public void clearArrayList() {
38          codici_errore.clear();
39      }
40
41      @Test
42      public void testCheckCredentials(){
43          assertEquals(codici_errore, checkCredentials("ser.dimartino@studenti.unina.it", "
                Password.123"));
44      }
45
46      @Test
47      public void tesPasswordVuota(){
48          codici_errore.add(9);
49          assertAll(
50                  () -> assertEquals(codici_errore, checkCredentials("fra.cutugno@studenti.
                        unina.it", null)),
51                  () -> assertEquals(codici_errore, checkCredentials("lu.starace@studenti.
                        unina.it", ""))
52          );
53      }
54
55      @Test
56      public void testPasswordTroppoCorta(){
57          codici_errore.add(10);
58          assertEquals(codici_errore, checkCredentials("gio.cutolo@studenti.unina.it", "
                Aa_09."));
59      }
60
61      @Test
62      public void testPasswordNonValida(){
63          codici_errore.add(11);
64          assertAll(
65                  () -> assertEquals(codici_errore, checkCredentials("alb.aloisio@studenti.
                        unina.it", "password?123")),  // MANCA LA MAIUSCOLA
66                  () -> assertEquals(codici_errore, checkCredentials("an.corazza@studenti.
                        unina.it", "PASSWORD#123")),  //MANCA LA MINUSCOLA
67                  () -> assertEquals(codici_errore, checkCredentials("alb.aloisio@studenti.
                        unina.it", "Password123")),  // MANCA IL CARATTERE SPECIALE
68                  () -> assertEquals(codici_errore, checkCredentials("an.corazza@studenti.
                        unina.it", "Password_unoduetre")) // MANCA IL NUMERO
69          );
70
71      }
72
```

```
73      @Test
74      public void testEmailVuota(){
75          codici_errore.add(12);
76          assertAll(
77                  () -> assertEquals(codici_errore, checkCredentials(null, "Password.123"))
                        ,
78                  () -> assertEquals(codici_errore, checkCredentials("", "Password.123"))
79          );
80      }
81
82      @Test
83      public void testEmaildNonValida(){
84          codici_errore.add(13);
85          assertAll(
86                  () -> assertEquals(codici_errore, checkCredentials("
                        emailcompletamentesbagliata", "Password.123")),  // EMAIL
                        COMPLETAMENTE SBAGLIATA
87                  () -> assertEquals(codici_errore, checkCredentials("@gmail.com", "
                        Password.123")),  // MANCA LO USERNAME
88                  () -> assertEquals(codici_errore, checkCredentials("biagio@.net", "
                        Password.123")),  // MANCA IL SECOND LEVEL DOMAIN
89                  () -> assertEquals(codici_errore, checkCredentials("matteo@libero.", "
                        Password.123")), // MANCA IL TOP LEVEL DOMAIN
90                  () -> assertEquals(codici_errore, checkCredentials("luigivirgilio.it", "
                        Password.123")), //MANCA LA @
91                  () -> assertEquals(codici_errore, checkCredentials("MATTEO[BIAGIO]
                        LUIGI_@libero.IT", "Password.123")) // LO USERNAME PRESENTA CARATTERI
                         NON CORRETTI
92          );
93      }
94
95      @Test
96      public void testErroriMultipli_9_12(){
97          codici_errore.add(9);
98          codici_errore.add(12);
99
100         assertEquals(codici_errore, checkCredentials("", ""));
101     }
102
103     @Test
104     public void testErroriMultipli_9_13(){
105         codici_errore.add(9);
106         codici_errore.add(13);
107
108         assertEquals(codici_errore, checkCredentials("@studenti.@libero@com", ""));
109     }
110
111     @Test
112     public void testErroriMultipli_10_12(){
```

```
113         codici_errore.add(10);
114         codici_errore.add(12);
115
116         assertEquals(codici_errore, checkCredentials("", "Ab.34"));
117     }
118
119     @Test
120     public void testErroriMultipli_10_13(){
121         codici_errore.add(10);
122         codici_errore.add(13);
123
124         assertEquals(codici_errore, checkCredentials("emailcompletamentesbagliata", "Ab
                .34"));
125     }
126
127     @Test
128     public void testErroriMultipli_11_12(){
129         codici_errore.add(11);
130         codici_errore.add(12);
131
132         assertEquals(codici_errore, checkCredentials("", "Password.Password"));
133     }
134
135     @Test
136     public void testErroriMultipli_11_13(){
137         codici_errore.add(11);
138         codici_errore.add(13);
139
140         assertEquals(codici_errore, checkCredentials("@studenti.@libero@com", "@studenti.
                it"));
141     }
142 }
```

2.2 Test funzione che calcola incasso in un range di date.

La funzione "getIncassoRangeGiorni" prende in input una data di inizio e una lista di ordini e calcola l'incasso totale degli ordini che sono stati effettuati tra la data di inizio e la data attuale.
La funzione controlla che la lista degli ordini non sia nulla. In caso contrario, restituisce zero. Successivamente, per ogni ordine nella lista degli ordini, la funzione controlla se la data dell'ordine è compresa tra la data di inizio e la data attuale. Se la data è compresa nel range, l'importo dell'ordine viene aggiunto all'incasso totale. Infine, la funzione restituisce l'incasso totale degli ordini effettuati nel range di tempo specificato.

Dato che la funzione "getIncassoRangeGiorni" fa parte della classe StatisticsActivity e fa uso della classe Ordini, abbiamo creato nel Package "Driver" i seguenti *Mock* per poter eseguire il testing:

```
144     public class StatisticsActivityMock {
145
146     public float media(int giorni, float incasso){
147         float media = 0;
148         if(giorni < 0) throw new IllegalArgumentException("Il numero di giorni non deve
                essere negativo");
149         if (giorni == 0) throw new ArithmeticException("Il numero di giorni non puo
                essere 0");
150         if (incasso < 0) throw new IllegalArgumentException("l'incasso deve essere
                maggiore di 0.");
151         media = incasso / giorni;
152         media = Math.round(media * 100) / 100f;
153
154         return media;
155     }
156
157     public int getIncassoRangeGiorni(LocalDate dataInizio, ArrayList<OrdineMock> orders){
158         int incassoTotale = 0;
159         LocalDate endDate = LocalDate.now();
160         DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd");
161
162         if (orders == null) return 0;
163         for (OrdineMock ordine : orders) {
164             LocalDate orderDate = LocalDate.parse(ordine.getDataOrdine(), formatter);
165             if (orderDate.isEqual(dataInizio) || orderDate.isAfter(dataInizio) &&
                    orderDate.isBefore(endDate)) {
166                 incassoTotale += ordine.getConto();
167
168
169             }
170         }
171         return incassoTotale;
172     }
173 }
```

```
174
175  /*
176     IN QUESTO CASO LA CLASSE ORDINE MOCK CONTIENE SOLO
177     GLI ATTRIBUTI DI ORDINE CHE ENTRANO IN GIOCO ALL'INTERNO
178     DEL METODO getIncassoRangeGiorni
179 */
180
181 public class OrdineMock {
182
183    private int conto;
184    private String dataOrdine;
185
186    public OrdineMock(int conto, String dataOrdine) {
```

```
187         this.conto = conto;
188         this.dataOrdine = dataOrdine;
189     }
190 
191     public int getConto() {
192         return conto;
193     }
194 
195     public String getDataOrdine() {
196         return dataOrdine;
197     }
198 }
```

```
200 
201 /*
202     CLASSI DI EQUIVALENZA:
203 
204         DATA_INIZIO: {VALIDA, NULL}
205           - Valida: a sua volta puo' essere
206                 1.1) Verosimile e precedente agli ordini
207                 1.2) Futura agli ordini
208                 1.3) Inverosimilmente precedente agli ordini
209 
210         ORDINI: {VALIDI, NULL, LISTA VUOTA, CON DATA SBAGLIATA}
211 
212  ------------------------------------------------------------------------------
213 
214     STRATEGIE DI TESTING UTILIZZATE:
215 
216         BlackBox secondo il criterio WECT
217 
218     CASI DI TESTING RITENUTI NECESSARI:
219       {VALIDA , VALIDI} : 1 Caso
220       {VALIDA , NULL} : 1 Caso
221       {VALIDA , LISTA VUOTA} : 1 Caso
222       {VALIDA , CON DATA SBAGLIATA} : 4 Casi
223       {DATA FUTURA , VALIDI} : 1 Caso
224       {DATA INVEROSIMILMENTE PRECEDENTE , VALIDI} : 1 Caso
225       {NULL , VALIDI} : 1 Caso
226 
227  ------------------------------------------------------------------------------ */
228 
229 public class getIncassoRangeGiorniTest {
230 
231     StatisticsActivityMock statisticsActivityMock;
232     ArrayList<OrdineMock> ordiniM;
233     DateTimeFormatter formatter;
234 
```

```
235     @Before
236     public void setUp(){
237         statisticsActivityMock = new StatisticsActivityMock();
238         ordiniM = new ArrayList<>();
239         formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd");
240     }
241
242     @AfterEach
243     public void clearArrayList(){
244         ordiniM.clear();
245     }
246
247
248     @Test
249     public void testGetIncassoRangeGiorni() {
250         ordiniM.add(new OrdineMock(3, "2023-05-04"));
251         ordiniM.add(new OrdineMock(105, "2023-05-04"));
252         ordiniM.add(new OrdineMock(72, "2023-02-04"));
253
254         LocalDate dataInizio = LocalDate.parse("2023-02-05", formatter);
255
256         int result = statisticsActivityMock.getIncassoRangeGiorni(dataInizio, ordiniM);
257         assertEquals(108, result);
258     }
259
260     @Test
261     public void testZeroOrdini()  {
262         LocalDate dataInizio = LocalDate.parse("2023-02-01", formatter);
263
264         int result = statisticsActivityMock.getIncassoRangeGiorni(dataInizio, ordiniM);
265         assertEquals(0, result);
266
267     }
268
269     @Test
270     public void testOrdiniNull()  {
271         LocalDate dataInizio = LocalDate.parse("2023-02-01", formatter);
272
273         int result = statisticsActivityMock.getIncassoRangeGiorni(dataInizio, null);
274         assertEquals(0, result);
275
276     }
277
278     @Test
279     public void testDataInizioNull()  {
280         ordiniM.add(new OrdineMock(3, "2023-05-04"));
281         ordiniM.add(new OrdineMock(105, "2023-05-04"));
282         ordiniM.add(new OrdineMock(72, "2023-02-04"));
283
```

```
284         assertThrows(NullPointerException.class,
285                 () -> statisticsActivityMock.getIncassoRangeGiorni(null, ordiniM));
286     }
287
288     @Test
289     public void testDataInizioFutura() {
290         ordiniM.add(new OrdineMock(3, "2023-05-04"));
291         ordiniM.add(new OrdineMock(105, "2023-05-04"));
292         ordiniM.add(new OrdineMock(72, "2023-02-04"));
293
294         LocalDate dataInizio = LocalDate.parse("2033-02-05", formatter);
295
296         int result = statisticsActivityMock.getIncassoRangeGiorni(dataInizio, ordiniM);
297         assertEquals(0, result);
298     }
299
300     @Test
301     public void testDataInizioIrrealistica() {
302         ordiniM.add(new OrdineMock(3, "2023-05-04"));
303         ordiniM.add(new OrdineMock(105, "2023-05-04"));
304         ordiniM.add(new OrdineMock(72, "2023-02-04"));
305
306         LocalDate dataInizio = LocalDate.parse("0133-02-05", formatter);
307
308         int result = statisticsActivityMock.getIncassoRangeGiorni(dataInizio, ordiniM);
309         assertEquals(180, result);
310     }
311
312     @Test
313     public void testDataOrdiniInFormatoSbagliato(){
314         ArrayList<OrdineMock> ordiniM_1 = new ArrayList<>();
315         ArrayList<OrdineMock> ordiniM_2 = new ArrayList<>();
316         ArrayList<OrdineMock> ordiniM_3 = new ArrayList<>();
317         ArrayList<OrdineMock> ordiniM_4 = new ArrayList<>();
318
319         ordiniM_1.add(new OrdineMock(3, "2023"));
320         ordiniM_2.add(new OrdineMock(105, "02/05/2023"));
321         ordiniM_3.add(new OrdineMock(72, "due-aprile-2023"));
322         ordiniM_4.add(new OrdineMock(105, "2023-32-31"));
323
324
325         LocalDate dataInizio = LocalDate.parse("2023-02-01", formatter);
326
327         assertAll(
328                 () ->   assertThrows(DateTimeParseException.class,
329                         () -> statisticsActivityMock.getIncassoRangeGiorni(dataInizio,
                                ordiniM_1)
330                 ),
331                 () ->   assertThrows(DateTimeParseException.class,
```

```
                        () ->statisticsActivityMock.getIncassoRangeGiorni(dataInizio,
                            ordiniM_2)
                ),
                () ->   assertThrows(DateTimeParseException.class,
                        () -> statisticsActivityMock.getIncassoRangeGiorni(dataInizio,
                            ordiniM_3)
                ),
                () ->   assertThrows(DateTimeException.class,
                        () -> statisticsActivityMock.getIncassoRangeGiorni(dataInizio,
                            ordiniM_4)
                )

        );
        ordiniM_1.clear();
        ordiniM_2.clear();
        ordiniM_3.clear();
        ordiniM_4.clear();
    }

}
```

2.3 Test funzione che controlla i campi di un ristorante.

Questo metodo prende in input quattro stringhe che rappresentano il nome di un ristorante, il numero di coperti, l'indirizzo' e il numero di telefono. Il metodo restituisce un oggetto ArrayList di interi che rappresenta una lista di eventuali errori verificatisi durante l'esecuzione del metodo.

```
public class getRestaurantFiedlsErrorsTest {
/*
    CLASSI DI EQUIVALENZA:

       NOME: {VALIDO, VUOTO, TROPPO CORTO}

       COPERTI: {VALIDO, VUOTO, FUORI-RANGE, NON VALIDO}
         - FUORI RANGE: <5 && > 1000
         - NON VALIDO: Non composto da soli numeri

       INDIRIZZO: {VALIDO, VUOTO, TROPPO CORTO, NON VALIDO}
         - NON VALIDO: Contiene caratteri speciali

       TELEFONO: {VALIDO, VUOTO, NON VALIDO}

  ------------------------------------------------------------------------------

    CODICI DI ERRORE:
     1 = NOME TROPPO CORTO (1 CARATTERE MINIMO)
     2 = NOME MANCANTE
     3 = NUMERO DI COPERTI MANCANTE
```

```
371      4 = NUMERO DI COPERTI FUORI RANGE
372      5 = INDIRIZZO MANCANTE
373      6 = INDIRIZZO TROPPO CORTA (MINIMO 5 CARATTERI)
374      7 = NUMERO DI TELEFONO MANCANTE
375      8 = NUMERO DI TELEFONO NON VALIDO (10 CIFRE NUMERICHE RICHIESTE)
376      9 = NUMERO DI COPERTI ERRATO
377      10 = INDIRIZZO NON VALIDO
378
379  ------------------------------------------------------------------------------------
380
381     STRATEGIE DI TESTING UTILIZZATE:
382        BlackBox secondo il criterio WECT
383
384     CASI DI TESTING RITENUTI NECESSARI:
385       {VALIDO, VALIDO, VALIDO, VALIDO} : 1 Caso
386       {TROPPO CORTO, VALIDO, VALIDO, VALIDO} : 1 Caso
387       {VALIDO, FUORI-RANGE, VALIDO, VALIDO} : 2 Casi
388       {VALIDO, NON VALIDO, VALIDO, VALIDO} : 2 Casi
389       {VALIDO, VALIDO, TROPPO CORTO, VALIDO} : 1 Caso
390       {VALIDO, VALIDO, NON VALIDO, VALIDO} : 1 Caso
391       {VALIDO, VALIDO, VALIDO, NON VALIDO} : 1 Caso
392       {NULL, NULL, NULL, NULL} : 1 Caso
393
394  ------------------------------------------------------------------------------ */
395     public ArrayList<Integer> codici_errore = new ArrayList<Integer>();
396
397
398     // L'ARRAYLIST DEVE ESSERE PULITO OGNI VOLTA CHE VIENE CONCLUSO UN CASO DI TEST
399     @AfterEach
400     public void clearArrayList(){
401         codici_errore.clear();
402     }
403
404     @Test
405     public void testgetRestaurantFieldsError(){
406         ArrayList<Integer> actualErrors = getRestaurantFieldsErrors("Ristorante Test", "
                10", "Via Roma 1", "0123456789");
407         assertEquals(codici_errore, actualErrors); //Funziona poiche non ci sono codici
                di errore: l'ArrayList risulta vuoto
408     }
409
410     @Test
411     public void testNomeCampoTroppoCorto() {
412         codici_errore.add(1);
413         ArrayList<Integer> actualErrors = getRestaurantFieldsErrors("a", "10", "Via Roma
                1", "0123456789");
414         assertEquals(codici_errore, actualErrors);
415     }
416
```

```
417     @Test
418     public void testCampoCopertiFuoriRange(){
419         codici_errore.add(4);
420         assertAll(
421                 () -> assertEquals(codici_errore, getRestaurantFieldsErrors("Ristorante
                        Test", "1", "Via Roma 1", "0123456789")),
422                 () -> assertEquals(codici_errore, getRestaurantFieldsErrors("Ristorante
                        Test", "100000", "Via Roma 1", "0123456789"))
423         );
424     }
425
426     @Test
427     public void testCampoCopertiNonValido(){
428         codici_errore.add(9);
429         assertAll(
430                 () -> assertEquals(codici_errore, getRestaurantFieldsErrors("Ristorante
                        Test", "dieci", "Via Roma 1", "0123456789")),
431                 () -> assertEquals(codici_errore, getRestaurantFieldsErrors("Ristorante
                        Test", "10a", "Via Roma 1", "0123456789"))
432         );
433     }
434
435
436     @Test
437     public void testLocazioneCampoTroppoCorto() {
438         codici_errore.add(6);
439         ArrayList<Integer> actualErrors = getRestaurantFieldsErrors("Ristorante Test", "
                10", "Via", "0123456789");
440         assertEquals(codici_errore, actualErrors);
441     }
442
443     @Test
444     public void testLocazioneNonValido() {
445         codici_errore.add(10);
446         ArrayList<Integer> actualErrors = getRestaurantFieldsErrors("Ristorante Test", "
                10", "Via_Napoli?", "0123456789");
447         assertEquals(codici_errore, actualErrors);
448     }
449
450
451     @Test
452     public void testNumeroTelefonoCampoTroppoCorto() {
453         codici_errore.add(8);
454         ArrayList<Integer> actualErrors = getRestaurantFieldsErrors("Ristorante Test", "
                10", "Via Roma 1", "12345678");
455         assertEquals(codici_errore, actualErrors);
456     }
457
458     @Test
```

```
459     public void testTuttiICampiVuoti() {
460         codici_errore.add(2);
461         codici_errore.add(3);
462         codici_errore.add(5);
463         codici_errore.add(7);
464         ArrayList<Integer> actualErrors = getRestaurantFieldsErrors("", "", "", "");
465         assertEquals(codici_errore, actualErrors);
466     }
467 }
```

2.4 Testing delle funzione media (in statistiche).

La funzione media prende in input il numero di giorni e l'incasso totale di un ristorante in quei giorni. Essa calcola la media giornaliera di incasso dividento l'incasso totale per il numero di giorni e restituisce il valore ottenuto. Inoltre, la funzione effettua alcune verifiche di validità sui parametri di input, come il controllo che il numero di giorni non sia negativo o pari a zero, e che l'incasso sia maggiore di zero. In caso di violazione di queste condizioni, la funzione lancia un'eccezione per segnalare l'errore.

```
468 public class mediaTest {
469 /*
470    CLASSI DI EQUIVALENZA:
471 
472       INCASSO: {NULL, NEGATIVO, POSITIVO}
473       GIORNI: {NULL, NEGATIVO, ZERO, POSITIVO}
474 
475  ------------------------------------------------------------------------------
476 
477    STRATEGIE DI TESTING UTILIZZATE:
478       BlackBox e WhiteBox secondo il criterio SECT
479 
480      ==== BLACKBOX ====
481 
482  ------------------------------------------------------------------------------ */
483 
484     StatisticsActivityMock mock;
485 
486     @Before
487     public void setUp(){
488         mock = new StatisticsActivityMock();
489     }
490 
491     @Test
492     public void testMedia(){
493         float result = mock.media(17, 50f);
494         assertEquals(2.94f, result, 0.001f);
495     }
496     @Test
497     public void testGiornoNegativoIncassoPositivo(){
```

```
498         assertThrows(IllegalArgumentException.class,
499                 () -> mock.media(-3, 50.06f)
500         );
501     }
502     @Test
503     public void testGiornoPositivoIncassoNegativo(){
504         assertThrows(IllegalArgumentException.class,
505                 () -> mock.media(3, -50.06f)
506         );
507     }
508     @Test
509     public void testGiornoEIncassoNegativi(){
510         assertThrows(IllegalArgumentException.class,
511                 () -> mock.media(-3, -50.06f)
512         );
513     }
514 
515     @Test
516     public void testGiornoZero(){
517         assertAll(
518                 () ->  assertThrows(ArithmeticException.class,
519                         () -> mock.media(0, 50.06f)
520                 ),
521                 () ->  assertThrows(ArithmeticException.class,
522                         () -> mock.media(0, -50.06f)
523                 )
524         );
525     }
```

```
526 /*      ==== WHITEKBOX ====
527 
528 ------------------------------------------------------------------------------ */
529 
530     @Test
531     public void testGiornoNull(){
532         Integer giorno = null;
533         assertAll(
534                 () ->   assertThrows(NullPointerException.class,
535                         () -> mock.media(giorno, 0.30f)
536                 ),
537                 () ->   assertThrows(NullPointerException.class,
538                         () -> mock.media(giorno, -0.30f)
539                 )
540         );
541     }
542     @Test
543     public void testIncassoNull(){
544         Float incasso = null;
```

```
545         assertAll(
546                 () ->   assertThrows(NullPointerException.class,
547                         () -> mock.media(0, incasso)
548                 ),
549                 () ->   assertThrows(NullPointerException.class,
550                         () -> mock.media(3, incasso)
551                 ),
552                 () ->   assertThrows(NullPointerException.class,
553                         () -> mock.media(-3, incasso)
554                 )
555         );
556     }
557
558     @Test
559     public void testCampiNull(){
560         Integer giorno = null;
561         Float incasso = null;
562         assertThrows(NullPointerException.class,
563                 () -> mock.media(giorno, incasso)
564         );
565     }
566
567
568 }
```

2.5 Usabilità sul campo

2.5.1 Premessa

Qualunque sia la tecnica utilizzata, i test con gli utenti sono indispensabili. Infatti, le cause delle difficoltà incontrate dagli utenti possono essere moltissime. Analizzare un sistema "a tavolino", come nelle valutazioni euristiche, anche se può permetterci d'individuare numerosi difetti, non è mai sufficiente. I problemi possono essere nascosti e verificarsi soltanto con certi utenti, in relazione alla loro esperienza o formazione. Cose ovvie per chi già conosce il sistema o sistemi analoghi possono rivelarsi difficoltà insormontabili per utenti meno esperti. Un test di usabilità ben condotto mette subito in evidenza queste difficoltà. La necessità del coinvolgimento degli utenti è affermata con chiarezza dalla stessa ISO 13407: " *La valutazione condotta soltanto da esperti, senza il coinvolgimento degli utenti, può essere veloce ed economica, e permettere di identificare i problemi maggiori, ma non basta a garantire il successo di un sistema interattivo. La valutazione basata sul coinvolgimento degli utenti permette di ottenere utili indicazioni in ogni fase della progettazione. Nelle fasi iniziali, gli utenti possono essere coinvolti nella valutazione di scenari d'uso, semplici mock-up cartacei o prototipi parziali. Quando le soluzioni di progetto sono più sviluppate, le valutazioni che coinvolgono l'utente si basano su versioni del sistema progressivamente più complete e concrete. Può anche essere utile una valutazione cooperativa, in cui il valutatore discute con l'utente i problemi rilevati.*"

2.5.2 Presentazione degli utenti

Per svolgere questo tipo di test importantissimi a prodotto finito abbiamo assunto che la nostra app fosse in versione "beta" e distribuito a una cerchia ristretta e selezionata di persone l'applicativo. Per avere un'idea dei test di valutazione dell'usabilià sul campo condotti riportiamo, tramite degli alias, quelle che sono state le "interviste" alle vere persone utilizzatrici. Gli intervistati sono ***Luigi, Biagio, Matteo e Massimo:***
Luigi, famoso imprenditore, voleva aprire un ristorante 3.0 in una zona turistica del proprio paese, e si è subito messo alla ricerca di un teami in grado di sostenerlo nella sua missione.
Matteo, conosciuto per essere il "capo" che chiunque desidera. Sa gestire gruppi di persone, è bravo a comunicare e a segnalare le criticà in ambienti lavorativi.
Biagio, gran lavoratore, ama darsi da fare e sfruttare il massimo dalle tecnlogie a disposizione per offrire un servizio sempre di qualità. Anni e anni di esperienza come cameriere in hotel e ristoranti fanno di lui la persona perfetta per far parte del team di Luigi.
Massimo, appena diciottenne, si è affacciato al mondo del lavoro e cerca la sua prima esperienza come addetto alla cucina. Scopre che il risotrante di Luigi cerca giovani in gamba!

2.5.3 Il confronto con gli utenti

Abbiamo dotato i dispositivi Android del personale del nuovissimo ristorante di Luigi della nostra applicazione Ratatuill23, e abbiamo monitorato per due giorni i risultati.
Domanda n.1: Luigi, è stato difficile registrarti e registrare il tuo ristorante nell'app?
Per nulla, anzi, la procedura di registrazione era guidata e un simpatico topolino mi ha guidato tra i vari controlli. Ho potuto abilitare l'opzione "zona turistica", ma a cosa serve?

Domanda n.2: Biagio, com'è stato prendere il tuo primo ordine da cellulare?
Molto seplice devo dire, non ho subito capito come rimuovere gli elementi dall'ordine, ma poi ho letto un'indicazione e ora so farlo! Inoltre improvvisamente il telefono ha suonato, era un avviso da Luigi!

Domanda n.3: Massimo, come ti sei trovato con la nostra app?
Effettivamente l'app è minimale e semplice da usare, in alcuni tratti forse un pochino troppo. In ogni caso, una notifica mi ha avvisato non appena è stato registrato un ordine.

Domanda n.4: Matteo, come ti sei trovato con la nostra app?
Come supervisore della mia sala, ad un certo punto ho avuto necessità di avvisare tutti i camerieri di un problema: grazie alla funzione avviso, ci sono riuscito facilmente.

Domanda n.5: Cosa migliorereste nella nostra applicazione?
Luigi: Avrei effettuato una scelta di colori diversa
Biagio: Forse avrei reso più intuitivo cancellare i piatti da un ordine.
Massimo:Non cambierei nulla, fa quello che deve fare, offrendo tutte le funzioni necessarie a una cucina.
Matteo: Sarebbe da incrementare la velocità di risposta in alcuni casi

Emerge da queste mini interviste un indice di gradimento dell'app di circa il 60/70%, da confrontare e confermare con l'uso nel tempo dell'applicativo.

2.5.4 Valutazione finale

Abbiamo notato che i feedback positivi sono stati maggiorni rispetto a quelli negativi e ciò rende il team entusiasta del proprio lavoro. Tra quelli negativi, notiamo che sono tutti risolvibili grazie alla scalabilità delle soluzioni tecnlogiche attivate e grazie alla possibilità di garantire aggiornamenti futuri in qualsiasi momento e su qualsiasi fronte del sistema.