

Software Testing - CMMTickets



**UNIVERSITÀ
DEGLI STUDI
DI BERGAMO**

Caprinali Michele 1087210
Mazzoleni Gabriel 1086530
Mazzoleni Raphael 1086531

Indice:

1. [Metodologia di Testing del Software](#)
2. [Codice di Test e Strumenti](#)
3. [Documentazione dei Test](#)
4. [Copertura dei Test](#)

1- Metodologia di Testing del Software

Nel nostro progetto, abbiamo cercato di adottare una strategia di testing orientata alla qualità, con l'obiettivo di garantire un'ampia copertura e una verifica approfondita del codice.

2- Codice di Test e Strumenti

I test sono stati effettuati sulle classi e metodi collegati al database per verificare l'assenza/presenza di bug o malfunzionamenti. Gli strumenti utilizzati per eseguire e implementare i casi di test sono stati i seguenti:

- *JUnit*: framework di unit testing per la programmazione in Java
- *SonarQube*: strumento di analisi statica del codice utile per identificare vulnerabilità, "code smell" e bug

3- Documentazione dei Test

Di seguito verranno mostrati dei casi di test sviluppati nell'applicazione CMMTickets, con la relativa spiegazione ed esecuzione.

A. Test per la classe EventsDatabase.java

```
@Test
@Order(1)
public void testGetAllEvents() throws SQLException {
    List<Evento> events = EventsDatabase.getAllEvents();
    assertNotNull(events);
}

@Test
@Order(2)
public void testAddEvento() throws SQLException {
    Evento evento = new Evento("Concerto", Date.valueOf("2025-01-30"), "20:00", 100, true, Date.valueOf("2025-01-29"), 400);
    boolean result = EventsDatabase.addEvento(evento);
    assertTrue(result);
}

@Test
@Order(3)
public void testUpdateEvento() throws SQLException {
    Evento updatedEvento = new Evento("Concerto aggiornato", Date.valueOf("2025-01-19"), "21:00", 150, false, Date.valueOf("2025-01-12"), 400);
    List<Evento> evs = EventsDatabase.getAllEvents();
    int id = evs.getLast().getIdEvento();
    boolean result = EventsDatabase.updateEvento(updatedEvento, id);
    assertTrue(result);
}

@Test
@Order(4)
public void testUpdateEventWithDefaultLuogo() throws SQLException {
    boolean result = EventsDatabase.updateEventWithDefaultLuogo(400);
    assertTrue(result);
}

@Test
@Order(5)
public void testDeleteEvento() throws SQLException {
    List<Evento> evs = EventsDatabase.getAllEvents();
    int id = evs.getLast().getIdEvento();
    boolean result = EventsDatabase.deleteEvento(id);
    assertTrue(result);
}
```

- a. *GetAllEvents* → verifica se il metodo funziona correttamente, recuperando tutti gli eventi dal database e controlla che la lista restituita non sia null.
- b. *AddEvento* → testa l'aggiunta di un nuovo evento al db, creando un nuovo oggetto con alcuni attributi e lo passa al metodo. Il test verifica che il risultato sia true, indicando che l'aggiunta è avvenuta con successo.
- c. *UpdateEvento* → testa l'aggiornamento di un evento già esistente, recuperando la lista degli eventi dal db e ottiene l'ID di uno di essi, crea un nuovo oggetto con attributi aggiornati e poi chiama il metodo, passando l'evento aggiornato e il relativo ID. Il test verifica che l'aggiornamento restituisca true.
- d. *UpdateEventWithDefaultLuogo* → testa una funzionalità specifica per aggiornare il luogo di un evento con un valore predefinito, chiamando il metodo, passando un ID (400) e verifica che restituisca true.
- e. *DeleteEvento* → testa la cancellazione di un evento, recuperando prima la lista degli eventi dal db e ottenendo l'ID, poi chiama il metodo con l'ID e verifica che la cancellazione restituisca true.

B. Test per la classe LuoghiDatabase.java

```
@Test
@Order(1)
public void testGetAllLuoghi() throws SQLException {
    List<Luogo> luoghi = LuoghiDatabase.getAllLuoghi();
    assertNotNull(luoghi);
}

@Test
@Order(2)
public void testAddLuogo() throws SQLException {
    Luogo luogo = new Luogo("Colosseo", "Roma", "Piazza del Colosseo", "colosseo.jpg");
    boolean result = LuoghiDatabase.addLuogo(luogo);
    assertTrue(result);
}

@Test
@Order(3)
public void testUpdateAndDeleteEvento() throws SQLException {
    Luogo updatedLuogo = new Luogo("Colosseo Aggiornato", "Roma", "Piazza del Colosseo", "colosseo.jpg");
    List<Luogo> evs = LuoghiDatabase.getAllLuoghi();
    int id= evs.getLast().getIdLuogo();
    boolean result = LuoghiDatabase.updateLuogo(updatedLuogo, id);
    assertTrue(result);
}

@Test
@Order(4)
public void testDeleteEvento() throws SQLException {
    Luogo updatedLuogo = new Luogo("Colosseo Aggiornato", "Roma", "Piazza del Colosseo", "colosseo.jpg");
    boolean result = LuoghiDatabase.deleteLuogo(updatedLuogo);
    assertTrue(result);
}
```

- a. *GetAllLuoghi* → verifica che il metodo della classe funzioni correttamente, recuperando tutti i luoghi dal db e controlla che la lista non sia null.
- b. *AddLuogo* → testa l'aggiunta di un nuovo luogo al db, creando un nuovo oggetto con attributi e lo passa al metodo, verificando che il risultato sia true, segno che l'aggiunta è riuscita.
- c. *UpdateAndDeleteEvento* → verifica l'aggiornamento di un luogo già esistente, recuperando la lista dei luoghi dal db e ottiene l'ID di uno. Poi crea un nuovo oggetto con le informazioni aggiornate e successivamente chiama il metodo passando il luogo aggiornato e il relativo ID. Controlla che il metodo restituito restituisca true.
- d. *DeleteEvento* → testa la cancellazione di un luogo dal db, creando un oggetto luogo. Passa il luogo al metodo per rimuoverlo dal db e verifica che il risultato sia true, indicando che la cancellazione è avvenuta con successo.

C. Test per la classe SectorsDatabase.java

```

int id;
@Test
@Order(1)
public void testGetAllSectors() throws SQLException, ParseException {
    List<Settore> sectors = SectorsDatabase.getAllSectors();
    assertNotNull(sectors);
}

@Test
@Order(2)
public void testAddSettore() throws SQLException {
    Settore settore = new Settore("Anello 1", 50.0f, "nord", 1, 200, 0, 1);
    boolean result = SectorsDatabase.addSettore(settore);
    assertTrue(result);
}

@Test
@Order(3)
public void testTicketAcquired() throws SQLException, ParseException {
    List<Settore> sectors = SectorsDatabase.getAllSectors();
    id= sectors.getLast().getIdSettore();
    boolean result = SectorsDatabase.ticketAcquired(id);
    assertTrue(result);
}

@Test
@Order(4)
public void testGetIdEvento() throws SQLException, ParseException {
    List<Settore> sectors = SectorsDatabase.getAllSectors();
    id= sectors.getLast().getIdSettore();
    int idEvento = SectorsDatabase.getIdEvento(id);
    assertNotEquals(0, idEvento);
}

@Test
@Order(5)
public void testDeleteSettori() throws SQLException, ParseException {
    List<Settore> sectors = SectorsDatabase.getAllSectors();
    id= sectors.getLast().getIdEvento();
    boolean result = SectorsDatabase.deleteSettori(id);
    assertTrue(result);
}

```

- a. *GetAllSectors* → verifica che il metodo restituisca una lista di settori valida, non deve essere null. Si assicura che il db riesca a recuperare correttamente i settori
- b. *AddSettore* → prova ad aggiungere un nuovo settore al db, restituisce true quando l'inserimento è avvenuto con successo.
- c. *TicketAcquired* → verifica che il metodo restituisca true, quando un determinato biglietto con l'ultimo ID della tabella settori è stato correttamente acquistato.
- d. *GetIdEvento* → Recupera l'ID dell'evento associato all'ultimo settore nella lista, passa il test se l'ID evento recuperato è diverso da 0. Verifica che ogni settore abbia un ID evento valido associato
- e. *testDeleteSettori* → tenta di eliminare un settore dal db, se il metodo restituisce true, indica che la cancellazione è avvenuta con successo. Controlla che sia possibile eliminare correttamente un settore dal db.

D. Test per la classe TicketDatabase.java







































```
@Test
void testAddTicket() {
    Biglietto biglietto = new Biglietto();
    biglietto.setNomeUtilizzatore("Giovanni");
    biglietto.setCognomeUtilizzatore("Rossi");
    biglietto.setPosto(10);
    biglietto.setIdSettore(2);
    biglietto.setIdUtente(1);
    boolean result = TicketsDatabase.addTicket(biglietto);
    assertTrue(result, "Il biglietto dovrebbe essere aggiunto con successo.");
}

@Test
void testGetAllUserTickets() {
    int idUser = 1;
    List<Biglietto> biglietti = TicketsDatabase.getAllUserTickets(idUser);
    assertNotNull(biglietti, "La lista dei biglietti non dovrebbe essere nulla.");
    assertTrue(biglietti.size() > 0, "L'utente dovrebbe avere almeno un biglietto.");
}
```

- a. *AddTicket* → testa che un nuovo biglietto venga aggiunto correttamente al sistema. Crea un nuovo oggetto e imposta i suoi attributi, chiama il metodo per aggiungere il biglietto al db e verifica che il metodo restituisca true, indicando che il biglietto è stato aggiunto con successo.
- b. *GetAllUserTickets* → lo scopo è che ci sia almeno un biglietto con l'utente ID (1) e che la lista non sia null. Viene specificato l'ID utente per il quale recuperare tutti i biglietti, viene chiamato il metodo e salvata la lista di biglietti, poi si controlla che la lista non sia null e che ci sia almeno un biglietto.

4- Copertura dei test

Di seguito viene mostrata la copertura dei test, nel nostro caso abbiamo testato le classi collegate al database per verificarne appunto l'effettivo funzionamento e che non fossero presenti bug/malfunzionamenti.

Element	Coverage	Covered Instru...	Missed Instruct...	Total Instructio...
▼  Progetto	 10,7 %	1.567	13.032	14.599
▼  src/main/java	 9,2 %	1.320	13.019	14.339
>  adminpanels_package	 0,0 %	0	6.006	6.006
>  userpanels_package	 0,0 %	0	4.645	4.645
>  login_package	 0,0 %	0	1.363	1.363
▼  database_package	 73,6 %	1.049	376	1.425
>  EventsDatabase.java	 76,0 %	383	121	504
>  SectorsDatabase.java	 73,9 %	298	105	403
>  LuoghiDatabase.java	 71,0 %	233	95	328
>  Database.java	 17,5 %	7	33	40
>  TicketsDatabase.java	 85,3 %	128	22	150
>  frames_package	 0,0 %	0	357	357
>  classes_package	50,8 %	271	262	533
>  utils_package	0,0 %	0	10	10
▼  src/test/java	95,0 %	247	13	260
>  ProgettoIngegneriaDelSoftware.	0,0 %	0	12	12
▼  test_package	 99,6 %	247	1	248
>  TestTicketDatabase.java	 97,8 %	45	1	46
>  TestEventsDatabase.java	 100,0 %	73	0	73
>  TestLuoghiDatabase.java	 100,0 %	58	0	58
>  TestSectorsDatabase.java	 100,0 %	71	0	71