

Università degli Studi di Salerno
Corso di Ingegneria del Software

BEAT – Booking Events And Tickets

Test Plan

Versione 1.2



Data: 25/11/2025

Coordinatore del progetto: Prof. Andrea De Lucia

Partecipanti: Carnevale Luigi[0512119029], Di Manso Carmine[0512119521], Clemente Manuel[0512119395]

Scritto da: Carnevale Luigi, Di Manso Carmine, Clemente Manuel

Repository GitHub: [github.com/Luigi-Carnevale/BEAT-Booking Events And Tickets](https://github.com/Luigi-Carnevale/BEAT-Booking-Events-And-Tickets)

Revision History

Data	Versione	Descrizione	Autore
25-11-2025	1.0	Creazione Test Plan	Carnevale Luigi, Di Manso Carmine, Clemente Manuel
21-12-2025	1.1	task 1	Carnevale Luigi
21-12-2025	1.2	completamento TASK 3	Di Manso Carmine

Indice

Revision History

1	Introduction	1
2	Relationship to other documents	2
2.1	Schema di naming e tracciabilità	2
2.2	Responsabilità del Task 1	2
3	System overview	3
3.1	Componenti testati (granularità unit)	3
3.2	Dipendenze principali	3
4	Features to be tested/not to be tested	4
4.1	Funzionalità da testare	4
4.2	Funzionalità non testate (con motivazione)	4
5	Pass/Fail criteria	5
5.1	Criteri generali	5
5.2	Criteri non funzionali (esempi di soglia)	5
5.3	Criteri specifici del Task 1	5
6	Approach	6
6.1	Strategia di integrazione	6
6.2	Tipologie di test	6
6.3	Gestione dipendenze esterne (mock/stub)	6
6.4	Focus del Task 1	7
7	Suspension and resumption	8
7.1	Criteri di sospensione	8
7.2	Criteri di ripresa	8
8	Testing materials (hardware/software requirements)	9
8.1	Requisiti hardware e Software	9
8.2	Risorse e strumenti speciali	9
9	Test cases	10
9.1	Suddivisione responsabilità (Task)	10
9.2	Elenco test case (high-level)	10
9.3	Documenti collegati	10
10	Testing schedule	11

1 Introduction

(Task 1 – Strategia di Test e Identity Management)

Questa sezione descrive gli obiettivi e l'estensione delle attività di testing previste per il sistema BEAT. Il Test Plan fornisce un riferimento operativo per pianificare ed eseguire i test in modo tempestivo e costo-efficiente, mantenendo la tracciabilità con requisiti (RAD) e decisioni di design/architettura (SDD).

Gli obiettivi principali del piano di test sono:

- verificare che le funzionalità implementate soddisfino i requisiti funzionali definiti nel RAD;
- verificare che i requisiti non funzionali più critici (prestazioni, disponibilità, sicurezza) siano rispettati entro soglie concordate;
- individuare difetti precocemente, riducendo il costo di correzione e il rischio di regressioni;
- definire responsabilità, ambienti, materiali e criteri di completamento dei test;
- validare con priorità la **porta di ingresso del sistema** (registrazione, login, ruoli, session management), da cui dipendono gli altri sottosistemi.

L'ambito include:

- **Unit test** di componenti e servizi applicativi;
- **Integration test** tra sottosistemi (es. Booking–Payment–Ticketing);
- **System test** end-to-end sulle principali user journey;
- **Security test** di base (hardening, input validation, session management) e controlli su vulnerabilità note (es. XSS/CSRF/SQLi);
- **Performance test** mirati sui flussi critici (acquisto biglietti, ricerca eventi).

2 Relationship to other documents

Il presente Test Plan è strettamente correlato ai documenti prodotti durante lo sviluppo:

- **RAD (Requirement Analysis Document):** definisce requisiti funzionali e non funzionali. Ogni caso di test viene tracciato rispetto a specifici requisiti (es. RF-xx, RNF-xx) e/o casi d'uso (UC_x).
- **SDD (System Design Document):** descrive architettura, sottosistemi e decisioni tecniche. I test di integrazione e di sistema verificano i flussi coerenti con la decomposizione architetturale.
- **ODD (Object Design Document):** descrive interfacce e contratti delle classi/servizi. Gli unit test validano precondizioni, postcondizioni ed eccezioni definite nelle interfacce.

2.1 Schema di naming e tracciabilità

Per garantire tracciabilità:

- Requisiti funzionali: **RF-xx** (o riferimento a **UC_x** nel RAD)
- Requisiti non funzionali: **RNF-xx**
- Test case: **TC-xx**
- Test incident: **TI-xx**

Esempio di corrispondenza: TC-02 valida UC_2 (Autenticazione) e contribuisce alla verifica dei requisiti di sicurezza e session management; TC-07 contribuisce alla verifica dei vincoli di consistenza e prevenzione overbooking (affidabilità transazionale).

2.2 Responsabilità del Task 1

(Task 1 – Strategia & Identity)

Il Task 1 utilizza i documenti RAD e SDD come fonte primaria per:

- definire la strategia di test complessiva (unit/integration/system) e i criteri generali di pass/fail;
- derivare i test su registrazione, autenticazione, gestione ruoli e controlli di sicurezza di base;
- impostare il modello di tracciabilità **Requisito/UC ↔ Test Case ↔ Incident**.

Le funzionalità relative a catalogo/recensioni e a transazioni/pagamenti sono trattate rispettivamente nel Task 2 e nel Task 3.

3 System overview

BEAT è un sistema web basato su architettura a tre livelli (Presentation, Application, Data). Ai fini del testing strutturale, i componenti principali che vengono testati in unit test e integrazione sono i seguenti:

3.1 Componenti testati (granularità unit)

- **User Management:** servizi di registrazione, autenticazione, gestione sessioni/ruoli.
- **Event Management:** servizi di creazione/modifica/cancellazione eventi, consultazione catalogo.
- **Booking & Ticketing:** logica prenotazione/acquisto, gestione disponibilità, emissione biglietti.
- **Payment Management:** gestione transazioni e interfaccia verso gateway esterno (mock/stub in test).
- **Notification:** composizione e invio notifiche (email); in test si usa un provider simulato.
- **Persistence/Data Management:** DAO/Repository e transazioni; in unit test si adottano DB embedded o test containers.

3.2 Dipendenze principali

- Booking dipende da Event (posti disponibili) e Payment (finalizzazione).
- User Management fornisce identità/ruoli a Event e Booking.
- Notification dipende dagli eventi di dominio generati da Booking/Payment/Event.
- Tutti i sottosistemi accedono ai dati attraverso il layer di persistenza.

4 Features to be tested/not to be tested

4.1 Funzionalità da testare

- **Autenticazione e autorizzazione:** login/logout, registrazione (cliente/organizzatore), gestione ruoli, access control matrix.
- **Catalogo eventi:** ricerca eventi per keyword/data/categoria, visualizzazione dettagli evento. [TASK 2 - TODO]
- **Gestione eventi (organizzatore/admin):** create/modify/delete, validazioni su data/ora/capienza. [TASK 2 - TODO]
- **Prenotazione e acquisto:** carrello, conferma ordine, prevenzione overbooking, consistenza ACID. [TASK 3 -]
- **Pagamento:** gestione esito transazione (successo/fallimento), rollback su errori. [TASK 3 -]
- **Biglietti:** generazione ticket con identificativo univoco (es. QR/ID), associazione all'utente. [TASK 3 -]
- **Notifiche:** invio conferme acquisto/prenotazione, notifiche annullamento/modifica evento. [TASK 3 -]
- **Sicurezza:** hashing password (output persistito non reversibile), protezioni base XSS/CSRF, SQL injection prevention (prepared statements). (Task 1)

4.2 Funzionalità non testate (con motivazione)

- **Gateway di pagamento reale:** in ambiente di test si usano sandbox/mock; l'integrazione reale dipende da credenziali e disponibilità del provider. [TASK 3 -]
- **Deliverability email end-to-end:** si valida la generazione del contenuto e l'invocazione del servizio; non si garantisce la consegna su tutti i provider reali.
- **Test su browser legacy:** si privilegiano browser moderni; eventuali compatibilità avanzate sono fuori ambito salvo requisiti espliciti.
- **Stampa fisica del titolo d'ingresso:** La verifica è limitata alla corretta generazione del file digitale (PDF e stringa QR Code). La compatibilità con dispositivi hardware di stampa non rientra negli obiettivi del software.
- **Resistenza ad attacchi di rete complessi (DDoS):** Test di vulnerabilità avanzati a livello di infrastruttura di rete o attacchi di negazione del servizio sono considerati fuori ambito per questa fase di test funzionale.

5 Pass/Fail criteria

I criteri di superamento/fallimento in questo piano sono generici e vengono raffinati nei documenti di specifica dei test.

5.1 Criteri generali

- Un test è **PASS** se tutti gli output osservati coincidono con l'oracolo di test (risultati attesi) e non si verificano eccezioni non previste.
- Un test è **FAIL** se almeno una asserzione fallisce, se si verifica un errore non gestito, o se non sono rispettate soglie prestazionali/sicurezza definite.

5.2 Criteri non funzionali (esempi di soglia)

- **Prestazioni:** le operazioni standard (caricamento pagine, login, ricerca, prenotazioni) devono mantenere un tempo medio di risposta inferiore a 2s in condizioni nominali.
- **Sicurezza:** password sempre memorizzate in forma hashata; input validati; assenza di SQLi in scenari base; gestione sessioni e controllo accessi coerenti con i ruoli.
- **Affidabilità transazionale:** in caso di errore durante acquisto/pagamento, nessuna prenotazione parziale deve rimanere persistita. [TASK 3 -]
- **Integrità dei dati (Zero Duplicati):** La soglia di errore per l'emissione di ticket duplicati per lo stesso posto deve essere lo 0%.
- **Rollback Success Rate:** Nel 100% dei casi di fallimento pagamento (TC-08), il sistema deve ripristinare correttamente la disponibilità dei posti entro **700ms**.

5.3 Criteri specifici del Task 1

(Task 1 – Identity & Security)

- La registrazione è **PASS** se l'email risulta univoca e la password non è mai memorizzata in chiaro (solo hash).
- L'autenticazione è **PASS** se la sessione viene creata correttamente e associata all'identità e al ruolo corretti.
- Ogni tentativo di accesso a funzionalità non autorizzate deve produrre un rifiuto lato server (403/redirect controllato) e non deve causare side-effect.
- Input malevoli (es. stringhe tipiche di SQLi/XSS) non devono produrre errori non gestiti né effetti persistenti (injection/alterazioni).

6 Approach

(Task 1 – Impostazione metodologica) L’approccio al testing combina test automatizzati e test manuali mirati, seguendo una strategia incrementale.

6.1 Strategia di integrazione

Si adotta un’integrazione **incrementale** basata su sottosistemi, con priorità ai flussi più critici:

- integrazione **User** → **Event**: accesso e consultazione eventi in base al ruolo;
- integrazione **Event** → **Booking**: verifica disponibilità e aggiornamento capienza;
- integrazione **Booking** → **Payment**: finalizzazione ordine e gestione esiti;
- integrazione **Booking/Payment** → **Notification**: invio conferme e notifiche.

6.2 Tipologie di test

- **Unit test**: su Service e componenti di dominio; dipendenze esterne mockate.
- **Integration test**: DB reale controllato (es. schema di test), servizi cooperanti, gateway/email simulati.
- **System test**: scenari end-to-end (utente reale → UI → backend → DB).
- **Regression test**: suite automatica eseguita ad ogni modifica significativa.
- **Security test (baseline)**: test su input malevoli, session handling, ruoli.

6.3 Gestione dipendenze esterne (mock/stub)

- Payment gateway: **mock** per stati successo/fallimento/timeouts. [TASK 3 -]
- Email provider: **stub** per verificare invocazione e contenuto.
- **Gateway di Pagamento ()**:
 - **Tecnica**: Mocking (tramite Mockito).
 - **Descrizione**: Si sostituisce l’interazione con l’API reale del provider con un oggetto "Mock" che restituisce risposte predefinite.
 - **Motivazione**: Permette di testare scenari critici (es. carta di credito rifiutata, timeout del server bancario) in modo sicuro, veloce e senza costi transazionali.
- **Servizio di Notifica (E-mail)**:
 - **Tecnica**: Stubbing.
 - **Descrizione**: Si utilizza una classe fittizia che implementa l’interfaccia `NotificationService` scrivendo l’esito dell’invio in un file di log locale.

- **Motivazione:** Consente di verificare che il sistema invochi correttamente l'invio del ticket dopo l'acquisto senza necessitare di un server SMTP reale o di una connessione internet.
- **Data Tier (Database):**
 - **Tecnica:** Fake Database (H2 o schema di test dedicato).
 - **Descrizione:** Uso di un database in-memory o di un'istanza MySQL "pulita" caricata con un dataset di test tramite file `.sql`.
 - **Motivazione:** Garantisce l'isolamento dei dati di test da quelli di produzione e assicura che ogni esecuzione parta da uno stato noto e controllato.

6.4 Focus del Task 1

Il Task 1 si concentra sulla validazione della **porta di ingresso** del sistema BEAT: registrazione, autenticazione, gestione sessioni e autorizzazione basata su ruoli. I test di integrazione iniziali verificano la corretta propagazione dell'identità e del ruolo verso i sottosistemi Event e Booking, così da assicurare che i vincoli di accesso siano applicati in modo consistente.

7 Suspension and resumption

[TASK 2 - TODO: definire metriche/criteri aggiuntivi se richiesti dal gruppo]

7.1 Criteri di sospensione

Le attività di test sui test item del piano possono essere sospese se:

- l'ambiente di test non è disponibile (DB down, configurazioni invalide);
- un difetto bloccante impedisce l'esecuzione di una parte significativa della suite;
- un cambiamento architetturale richiede riallineamento delle specifiche di test.

7.2 Criteri di ripresa

Quando i test riprendono:

- si ripetono i test di smoke (sanity) per verificare stabilità dell'ambiente;
- si riesegue la suite di regressione sulle aree impattate;
- si rieseguono i test falliti associati agli incident report aperti.

8 Testing materials (hardware/software requirements)

[TASK 3 :]

Questa sezione elenca le risorse necessarie per eseguire i test.

8.1 Requisiti hardware e Software

- Per l'esecuzione dei test di sistema e di integrazione del modulo Booking & Payment, sono necessari i seguenti requisiti:
 - **Hardware: Workstation di test:** Processore con architettura x64 (min. 4 core), 8 GB di RAM (consigliati 16 GB per l'esecuzione contemporanea di IDE, DBMS e Browser), 240GB spazio libero su disco (lo spazio necessario é molto inferiore ma il database potrebbe diventare importante).
 - **Software:**
 - * **Ambiente di esecuzione:** Java Development Kit (JDK) versione 17 o superiore.
 - * **Database:** MySQL Server 8.0 per la persistenza dei dati di test.
 - * **Build Tool:** Apache Maven per la gestione delle dipendenze e l'automazione del build.
 - * **OS:** Windows 10/11, macOS o distribuzioni Linux (Ubuntu 20.04+).

8.2 Risorse e strumenti speciali

- Dataset di test: utenti, eventi, prenotazioni pre-caricate.
- Log centralizzati (anche locali) per diagnosi incidenti.
- Script di reset DB per ripristino rapido dello stato.

Per la verifica della logica transazionale e dei pagamenti, verranno utilizzati i seguenti strumenti specifici:

- **JUnit 5:** Framework principale per la scrittura ed esecuzione degli unit test sulle classi di business (`BookingService`, `PaymentService`).
- **Mockito:** Essenziale per effettuare il **Mocking** del gateway di pagamento esterno, permettendo di simulare risposte di successo o fallimento senza effettuare transazioni reali.
- **Postman:** Utilizzato per il testing delle API REST (Endpoint del `BookingController`) inviando payload JSON e verificando i codici di stato HTTP (200 OK, 400 Bad Request, ecc.).

9 Test cases

Questa sezione elenca i test case previsti. Ogni test case è descritto in dettaglio in un documento separato di *Test Case Specification*. Ogni esecuzione dei test viene documentata tramite *Test Incident Report* quando emergono anomalie.

9.1 Suddivisione responsabilità (Task)

- **Task 1 ():** TC-01, TC-02, TC-09 (+ sicurezza di base su input/sessioni/ruoli).
- **Task 2 ():** TC-03, TC-04, TC-05, (TC-Recensioni se presente).
- **Task 3 ():** TC-06, TC-07, TC-08, TC-10 (notifiche) e test transazionali/ACID.

9.2 Elenco test case (high-level)

9.3 Documenti collegati

- **Test Case Specification:** per ogni TC-xx contiene precondizioni, passi, dati, expected results e criteri di pass/fail specifici.
- **Test Incident Report:** per ogni difetto rilevato contiene ID incidente, descrizione, severità, passi per riprodurre, log e stato.

10 Testing schedule

[TASK 3 : aggiornare schedule se il team ha milestone diverse]

La pianificazione del testing segue lo sviluppo incrementale. Le date sono indicative e vengono aggiornate in base all'avanzamento.

ID	Feature	Riferimenti	Descrizione sintetica
TC-01	Registrazione cliente	UC_1, RNF-Security	Creazione utente con email unica e password memorizzata solo in forma hashata.
TC-02	Login	UC_2, RNF-Security	Autenticazione con credenziali valide/invalidhe e corretta gestione sessione.
TC-03	Ricerca eventi	UC_4	Ricerca per keyword/data/categoria con risultati coerenti. [TASK 2 - TODO]
TC-04	Dettaglio evento	UC_4	Visualizzazione dettagli (posti, prezzo, luogo, data/ora). [TASK 2 - TODO]
TC-05	Creazione evento	UC_7	Organizzatore crea evento con vincoli corretti su data/capienza. [TASK 2 - TODO]
TC-06	Acquisto biglietto (successo)	UC_3, RNF-Performance	Validazione del flusso completo: verifica disponibilità, esito positivo del pagamento e generazione del titolo d'ingresso.
TC-07	Acquisto biglietto (sold out)	UC_3	Tentativo di acquisto su ultimo posto: verifica gestione della concorrenza e prevenzione overbooking.
TC-08	Pagamento fallito & rollback	UC_3, ACID	Fallimento transazione: verifica del ripristino della disponibilità posti e assenza di ticket nel DB.
TC-09	Access control matrix	SDD-Access Control	Operazioni consentite/negate per ruolo (cliente/organizzatore/admin).
TC-10	Notifiche conferma	UC_3/UC_7	Generazione corretta del ticket con QR Code e simulazione invio e-mail tramite provider stub.

Fase	Periodo	Responsabile	Output atteso
Setup ambiente test	Settimana 1	Team	DB test, dataset, pipeline test/coverage.
Unit test core services	Settimane 1–2	Team	Suite unit completa su User/Event/Booking.
Integration test (subsystem)	Settimane 2–3	Team	Integrazione Booking–Payment–Notification.
System test end-to-end	Settimana 3	Team	Scenari principali utente/organizzatore verificati.
Performance test mirati	Settimana 3	Team	Misure su flussi critici e report risultati.
Regression & hardening	Settimana 4	Team	Stabilizzazione, fix difetti, riesecuzione suite.