Università degli Studi di Salerno
Corso di Ingegneria del Software

AniTour
Test Case Specification
Versione 1.1

Data: 22/12/2025

Progetto: AniTour	Versione: 1.1
Documento: Test Case Specification	Data: 22/12/2025

Indice

0. Test Case Derivation (Category Partition Method)

Questa sezione preliminare documenta l'analisi svolta per derivare i valori di input della sezione 3 ("Input specifications").

Per la funzionalità `processCheckout` del sottosistema Booking, sono state identificate le seguenti categorie e scelte:

Categoria	Scelta (Choice)	ID	Vincolo
Stato carrello	Vuoto	C1	[Error]
	Contiene tour disponibile	C2	[Valid]
	Contiene tour esaurito	C3	[Error]
Pagamento	Carta valida	P1	[Valid]
	Carta rifiutata	P2	[Error]
Autenticazione	Utente loggato	U1	[Valid]

1. Test Case 01: Acquisto con successo (Happy path)

1.1. Test case specification identifier TC-BOOK-01

Checkout di un Tour disponibile con pagamento valido.

1.2. Test items

- **Componente:** `BookingControl`
- **Feature:** `processCheckout(sessionToken, paymentData)`
- **Obiettivo:** verificare che una transazione valida crei un ordine confermato e aggiorni lo stock.

1.3. Input specifications

- **SessionToken:** "VALID_SESSION_USER_1" (Corrisponde alla scelta U1 + C2).
  - **Contesto:** il carrello associato contiene 2 biglietti per il Tour "Persona 5".
  - **Stato database:** il Tour "Persona 5" ha 50 posti disponibili.
- **PaymentData:** oggetto DTO con numero carta "4000-0000-0000-0000" (Corrisponde alla scelta P1).

1.4. Output specifications

- **Valore di ritorno:** oggetto `Booking` con `status = CONFIRMED` e `id > 0`
- **Stato del sistema (Post-condizioni):**
  1. La tabella `bookings` contiene un nuovo record collegato all'utente.
  2. La disponibilità del Tour "Persona 5" nel DB è scesa a 48 (50 – 2).
  3. Il carrello associato alla sessione è vuoto.

1.5. Environmental needs

- **Hardware:** PC di sviluppo standard.
- **Software:** Java 17, JUnit 5.
- **Test drivers:** JUnit Test Runner per invocare il metodo `processCheckout.`
- **Stubs/Mocks:**
  - `MockPaymentGateway`: configurato per restituire `true` (Autorizzato) alla chiamata `processPayment`.
  - Database di Test MySQL con dati pre-caricati.

1.6. Special procedural requirements

- **Setup:** prima del test, il database deve essere pulito e popolato con il record del Tour "Persona 5" (50 posti).
- **Teardown:** al termine, i dati di test devono essere rimossi per non influenzare i test successivi (rollback automatico).

1.7. Intercase dependencies

Nessuna.

2. Test Case 02: Fallimento per posti esauriti (Concurrency)

2.1. Test case specification identifier TC-BOOK-02

Tentativo di checkout su Tour divenuto sold out.

2.2. Test items

- **Componente:** `BookingControl`
- **Feature:** Gestione eccezioni in `processCheckout`
- **Obiettivo:** verificare che il sistema impedisca la creazione dell'ordine e non decrementi lo stock se il tour richiesto ha posti disponibili pari a zero (gestione concorrenza/sold Out).

2.3. Input specifications

- **SessionToken:** "VALID_SESSION_USER_1" (Corrisponde alla scelta U1 + C3).

    - **Contesto:** il carrello associato contiene 1 biglietto per il Tour “Bloodborne”.
    - **Stato database:** il Tour “Bloodborne” ha 0 posti disponibili.
- **PaymentData:** carta valida "4000-0000-0000-0000" (P1).

2.4. Output specifications

- **Valore di ritorno:** Il metodo deve lanciare l'eccezione `SoldOutException`.
- **Stato del sistema (Post-condizioni):**
    1. Nessun ordine creato nella tabella `bookings`.
    2. Il carrello non viene svuotato (l'utente può rimuovere l'item).

2.5. Environmental needs

- Vedi specifica *TC-BOOK-01* (l’ambiente è condiviso).
- **Stubs/Mocks:**
    - `MockBookingRepository`: Configurato per simulare che il metodo `checkAvailability` restituisca `false`.

2.6. Special procedural requirements

Nessuna.

2.7. Intercase dependencies

Nessuna.

3. Test Case 03: Pagamento rifiutato

3.1. Test case specification identifier TC-BOOK-03

Gestione del rifiuto transazione da parte della banca.

3.2. Test items

- **Componente:** `BookingControl`
- **Feature:** Integrazione con `PaymentGateway`.
- **Obiettivo:** assicurarsi che nessuna prenotazione venga salvata nel db se il servizio di pagamento esterno (`PaymentGateway`) restituisce un esito negativo (es. carta rifiutata).

3.3. Input specifications

- **SessionToken:** “VALID_SESSION_USER_1” (U1 + C2).
- **PaymentData:** numero carta " 4000-9999-9999-9999" (Specifica per P2: fondi insufficienti).

3.4. Output specifications

- **Valore di ritorno:** Il metodo deve lanciare l'eccezione `PaymentFailedException.`
- **Stato del sistema (Post-condizioni):**
    1. Nessun ordine salvato nel db.
    2. Disponibilità posti invariata.

3.5. Environmental needs

- Vedi specifica *TC-BOOK-01* (l'ambiente è condiviso).
- **Stubs/Mocks:**
    - `MockPaymentGateway:` Configurato esplicitamente per restituire `false` quando riceve il numero di carta terminante in "9999" .

3.6. Special procedural requirements

Nessuna.

3.7. Intercase dependencies

Nessuna.

4. Test Case 04: Carrello vuoto

4.1. Test case specification identifier TC-BOOK-04

Tentativo di checkout con carrello vuoto.

4.2. Test items

- **Componente:** `BookingControl`
- **Feature:** validazione pre-condizioni in `processCheckout.`
- **Obiettivo:** verificare che il meccanismo di validazione intercetti le pre-condizioni invalide (lista articoli vuota) lanciando l'eccezione appropriata prima di tentare qualsiasi interazione con il db o il sistema di pagamento.

4.3. Input specifications

- **SessionToken:** "VALID_SESSION_EMPTY" (C1).
    - **Contesto:** esiste una sessione attiva, ma la lista degli elementi nel carrello ha `size() == 0`.
- **PaymentData:** dati carta validi o nulli (irrilevante).

4.4. Output specifications

- **Valore di ritorno:** Il metodo deve lanciare l'eccezione `EmptyCartException` (o `IllegalArgumentException`).
- **Stato del sistema (Post-condizioni):**
  1. Nessun ordine creato nel db.
  2. Nessuna chiamata effettuata al `PaymentGateway`.

4.5. Environmental needs

- Vedi specifica *TC-BOOK-01* (l'ambiente è condiviso).
- **Stubs/Mocks:**
  - `MockCartManager`: configurato per restituire una lista vuota quando interrogato.

4.6. Special procedural requirements

Nessuna.

4.7. Intercase dependencies

Nessuna.

Revision History

Data	Versione	Descrizione	Autore
22/12/2025	1.0	Creazione documento Test Case Specification	Vincenzo Chiocca
22/12/2025	1.1	Aggiunto TC-BOOK-04 mancante al documento. Aggiunti obiettivi mancanti ai test case.	Vincenzo Chiocca

Partecipanti:

Nome	Matricola

Vincenzo Chiocca	**0512119182**
Salvatore Merola	**0512120979**