Università degli Studi di Salerno
Corso di Ingegneria del Software

Test Cases Document - HeavyRoute

Versione 1.0

Informazioni Generali

Corso di Laurea:	Informatica
Università:	Università degli Studi di Salerno (UNISA)
Docente:	Chiar.mo Prof. DE LUCIA Andrea
Data:	22/12/2025
Anno Accademico:	2025/2026

Membri Del Gruppo

MANFREDINI Umberto	Matricola 0512119797
MANZO Ugo	Matricola 0512119071 *(Coordinatore)*
ROMANO Pino Fiorello	Matricola 0512120259

Revision History

Data	Versione	Descrizione	Autore
22/12/2025	1.0	Creazione del Test Case Doc.	Pino Fiorello Romano
–	–	–	–
–	–	–	–

Indice

1. Test Case 1: Creazione Richiesta Valida

1.1. 1. Test case specification identifier

TC-CORE-01

1.2. 2. Test items

- **Unit Under Test (UUT):** Classe `ViaggioService`.
- **Method:** `creaRichiesta(RichiestaDTO dto, Long clientId)`.
- **Feature:** Creazione e persistenza di una nuova richiesta (UC3).

1.3. 3. Input specifications

Argomenti del metodo:

- `dto`: Un oggetto `RichiestaDTO` popolato come segue:

```
{
    "origine": "Napoli, Via Roma 1",
    "destinazione": "Roma, Via Tiburtina 10",
    "tipoCarico": "Macchinari Industriali",
    "peso": 1500.0,
    "dataRitiro": [Data futura, es. 2026-06-30]
}
```

- `clientId`: `1L`.

Stato del Mock:

- `UserRepository` deve restituire un oggetto `User` valido quando interrogato con ID 1.
- `RichiestaRepository` deve restituire l'oggetto salvato quando viene invocato `save()`.

1.4. 4. Output specifications (Oracle)

Il test è considerato superato se:

1. Il metodo restituisce un oggetto `RichiestaTrasporto` non nullo.
2. `oggetto.getId()` non è nullo (simulazione persistenza).

3. `oggetto.getStato()` è uguale a `IN_ATTESA`.
4. `oggetto.getCommittente().getId()` è uguale a 1.
5. Viene invocato esattamente una volta il metodo `save()` sul mock del repository.

1.5. 5. Environmental needs

- **Framework:** JUnit 5.
- **Driver:** Test runner standard di JUnit.
- **Stubs/Mocks:** `Mockito` per `UserRepository` e `RichiestaRepository`.

1.6. 6. Special procedural requirements

Configurare i mock (`when(...).thenReturn(...)`) prima dell'esecuzione del metodo.

1.7. 7. Intercase dependencies

Nessuna.

2. Test Case 2: Validazione Input Richiesta

2.1. 1. Test case specification identifier

TC-CORE-02

2.2. 2. Test items

- **Unit Under Test (UUT):** `ViaggioController` (in contesto Spring).
- **Endpoint:** `POST /api/requests`.
- **Feature:** Validazione automatica del payload JSON tramite annotazioni (Bean Validation).

2.3. 3. Input specifications

Richiesta HTTP Simulata:

- **Method:** POST
- **Content-Type:** application/json
- **Payload (JSON Invalido):**

```
{
    "origine": "",
    "destinazione": "Roma",
    "peso": -50.0,
    "dataRitiro": "2020-01-01" // Errore, Data passata
}
```

2.4. 4. Output specifications (Oracle)

Il test è considerato superato se la risposta HTTP soddisfa i seguenti criteri:

1. **Status Code:** `400 Bad Request`.
2. **Content-Type:** `application/problem+json`.
3. **Body:** Contiene messaggi di errore specifici per i campi invalidi (es. "must not be blank", "must be greater than 0", "must be a future date").
4. **Verifica Service:** Il metodo `ViaggioService.creaRichiesta` NON è stato invocato (verifica tramite `verifyNoInteractions(serviceMock)`).

2.5. 5. Environmental needs

- **Framework:** Spring Boot Test.
- **Driver:** `MockMvc` per inviare la richiesta e asserire sulla risposta.
- **Stubs:** Il Service layer è mockato per isolare il test del Controller.

2.6. 6. Special procedural requirements

Configurare l'ambiente di test con `@WebMvcTest(ViaggioController.class)` per caricare solo il layer web, ottimizzando le prestazioni.

2.7. 7. Intercase dependencies

Nessuna.

3. Test Case 3: Approvazione Richiesta e Generazione Viaggio

3.1. 1. Test case specification identifier

TC-CORE-03

3.2. 2. Test items

- **Unit Under Test (UUT):** `ViaggioService`.
- **Method:** `approvaEGeneraViaggio(Long requestId)`.
- **Feature:** Approvazione logistica e istanziazione del Viaggio (UC4).

3.3. 3. Input specifications

- **Argomento:** `requestId = 10L`.
- **Stato Iniziale (Precondizione):**
  - Nel repository esiste una `RichiestaTrasporto` con ID 10.
  - Lo stato della richiesta è `IN_ATTESA`.

3.4. 4. Output specifications (Oracle)

Il test è considerato superato se:

1. **Valore di Ritorno:** Un oggetto `Viaggio` non nullo.
2. **Stato Viaggio:** `IN_PIANIFICAZIONE`.
3. **Relazione:** `viaggio.getRichiesta().getId()` è uguale a 10.
4. **Side Effect (Richiesta):** Recuperando nuovamente la richiesta ID 10 dal DB, il suo stato è ora `APPROVATA`.
5. **Side Effect (Persistenza):** Il metodo `tripRepository.save()` è stato invocato.

3.5. 5. Environmental needs

- **Framework:** JUnit 5.
- **Database:** H2 in-memory (per testare la persistenza reale delle entità e le FK) oppure Mockito (per testare solo la logica in isolamento).

3.6. 6. Special procedural requirements

Verificare la transazionalità: se il salvataggio del Viaggio fallisce (es. eccezione simulata nel repository), lo stato della Richiesta deve tornare a `IN_ATTESA` (Rollback).

3.7. 7. Intercase dependencies

Nessuna dipendenza di esecuzione (il dato di input è un mock), ma logicamente segue la creazione della richiesta (TC-CORE-01).

4. Test Case 4: Assegnazione Risorse a un Viaggio

4.1. 1. Test case specification identifier

TC-CORE-04

4.2. 2. Test items

- **Unit Under Test (UUT):** `ViaggioService`.
- **Method:** `assegnaRisorse(Long tripId, Long driverId, Long vehicleId)`.
- **Feature:** Associazione di Autista e Veicolo a un Viaggio pianificato (UC4).

4.3. 3. Input specifications

- **Argomenti:** `tripId=50L`, `driverId=200L`, `vehicleId=300L`.
- **Stato Iniziale (Mock DB):** Esiste un Viaggio ID 50 in stato `IN_PIANIFICAZIONE`.
- **Stato Iniziale (Mock ResourceService):**
  - `isDriverAvailable(200L, ...)` restituisce `true`.
  - `isVehicleAvailable(300L, ...)` restituisce `true`.

4.4. 4. Output specifications (Oracle)

Il test è considerato superato se:

1. Il metodo viene eseguito senza lanciare eccezioni.
2. L'oggetto `Viaggio` recuperato dal repository ha ora:
   - `getDriver().getId() == 200L`
   - `getVehicle().getId() == 300L`
3. **Verifica Interazioni:** Viene verificato che i metodi di controllo disponibilità su `ResourceService` siano stati invocati esattamente una volta con i parametri corretti.

4.5. 5. Environmental needs

- **Framework:** JUnit 5.
- **Mocks:** `Mockito` è essenziale per simulare il comportamento del `ResourceService` e isolare il test dalla logica di gestione risorse.

4.6. 6. Special procedural requirements

Configurare i mock (`when(...).thenReturn(true)`) prima dell'esecuzione per simulare uno scenario "happy path" di disponibilità.

4.7. 7. Intercase dependencies

Logicamente dipendente dall'esistenza di un viaggio (TC-CORE-03).

5. Test Case 5: Gestione Conflitto Risorse (Risorsa Occupata)

5.1. 1. Test case specification identifier

TC-CORE-05

5.2. 2. Test items

- **Unit Under Test (UUT):** `ViaggioService`.
- **Method:** `assegnaRisorse(...)`.
- **Feature:** Gestione delle eccezioni di business e prevenzione dei conflitti di allocazione (NFR Affidabilità).

5.3. 3. Input specifications

- **Argomenti:** `tripId=50L`, `driverId=200L`, `vehicleId=300L` (Stessi di TC-04).
- **Stato Iniziale (Mock ResourceService):**
  - `isDriverAvailable(200L)` restituisce `true`.
  - `isVehicleAvailable(300L)` restituisce **`false`** (simulazione risorsa occupata).

5.4. 4. Output specifications (Oracle)

Il test è considerato superato se:

1. Il metodo lancia un'eccezione di tipo `ResourceNotAvailableException`.
2. Il messaggio dell'eccezione indica chiaramente quale risorsa è indisponibile (es. "Veicolo 300 non disponibile").
3. **Verifica Stato:** L'oggetto `Viaggio` con ID 50 nel repository **NON è stato modificato**. I campi driver e vehicle devono rimanere `null` (o al valore precedente).
4. Il metodo `tripRepository.save()` NON è stato invocato (o è stato fatto rollback).

5.5. 5. Environmental needs

- **Framework:** JUnit 5 + Mockito.
- **Configurazione:** È necessario utilizzare `assertThrows` di JUnit per catturare e verificare l'eccezione.

5.6. 6. Special procedural requirements

Configurare i mock in modo asimmetrico (uno disponibile, uno no) per verificare che il controllo sia completo su tutte le risorse.

5.7. 7. Intercase dependencies

TC-CORE-04 (è il caso negativo corrispondente).

6. Test Case 6: Sicurezza - Accesso Non Autorizzato

6.1. 1. Test case specification identifier

TC-CORE-06

6.2. 2. Test items

- **Unit Under Test (UUT):** `ViaggioController` (Security Filter Chain).
- **Endpoint:** `POST /api/trips/10/approve`.
- **Feature:** Verifica delle regole di autorizzazione RBAC (Role-Based Access Control).

6.3. 3. Input specifications

Contesto di Sicurezza Simulato:

- **Utente:** "user_committente".
- **Ruoli:** `["ROLE_COMMITTENTE"]`.

Richiesta HTTP:

- **Method:** POST
- **URL:** `/api/trips/10/approve`

6.4. 4. Output specifications (Oracle)

Il test è considerato superato se:

1. La risposta HTTP ha lo status code **403 Forbidden**.
2. **Verifica Isolamento:** Il metodo sottostante `ViaggioService.approveRequest(10L)` **NON** è stato assolutamente invocato (verifica con `verifyNoInteractions(serviceMock)`).

6.5. 5. Environmental needs

- **Framework:** Spring Boot Test + Spring Security Test.
- **Driver:** `MockMvc` per l'invocazione dell'endpoint.

6.6. 6. Special procedural requirements

Utilizzare l'annotazione `@WithMockUser(username="test", roles={"COMMITTENTE"})` per iniettare il contesto di sicurezza prima dell'esecuzione della richiesta, simulando un utente loggato ma non autorizzato.

6.7. 7. Intercase dependencies

Nessuna.