Università degli Studi di Salerno
Corso di Ingegneria del Software

Test Plan Document - HeavyRoute

Versione 1.2

Informazioni Generali

Corso di Laurea:	Informatica
Università:	Università degli Studi di Salerno (UNISA)
Docente:	Chiar.mo Prof. DE LUCIA Andrea
Data:	13/01/2026
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
20/12/2025	1.0	Creazione del Test Plan Doc.	Umberto Manfredini
29/12/2025	1.1	Aggiunta Category Partition	Pino Fiorello Romano
13/01/2026	1.2	Aggiornamento Feature e Naming Convention	Pino Fiorello Romano

Indice

1. Introduction

1.1. Objectives

Il presente documento costituisce il Test Plan per il sottosistema *Core Business* del progetto *HeavyRoute*. Gli obiettivi primari di questo piano sono:

- Definire la strategia globale di verifica e validazione (V&V) per garantire che il software soddisfi i requisiti di qualità attesi.
- Identificare le risorse umane, hardware e software necessarie per l'esecuzione dei test.
- Stabilire un calendario chiaro per le attività di testing, integrato con il piano di sviluppo.
- Fornire una guida per i tester e gli sviluppatori coinvolti nel processo.

1.2. Scope

Il piano copre le attività di **Unit Testing** (verifica dei singoli componenti) e **Integration Testing** (verifica delle interazioni tra componenti) limitatamente al sottosistema *Core Business*. Sono esclusi da questo piano specifico i test di sistema end-to-end, i test di carico e i test di accettazione utente.

2. Relationship to other documents

Questo Test Plan funge da documento di coordinamento e mantiene relazioni di tracciabilità con i seguenti artefatti del progetto:

- **Requirements Analysis Document (RAD):** Fonte primaria dei requisiti.
  - I test di integrazione verificano la corretta implementazione dei *Functional Requirements (FR)* e dei casi d'uso descritti nel RAD.
  - I test verificano anche il rispetto dei requisiti non funzionali, come la validazione dei dati.
- **Object Design Document (ODD):** Fonte delle specifiche tecniche.
  - I test unitari sono derivati direttamente dalle interfacce delle classi e dai contratti (pre e post-condizioni) definiti nell'ODD.
- **Test Case Specification (TCS):** Documento operativo.

– Mentre il Test Plan definisce *cosa* testare e *quando*, il TCS contiene la specifica tecnica dettagliata di come eseguire ogni singolo test case (Input, Procedura, Oracolo/Output Atteso).

3. System overview

Il sistema oggetto di test è il backend del modulo `Core Business`, sviluppato con il framework **Spring Boot**. L'architettura segue il pattern a 3 livelli (Controller-Service-Repository).

3.1. Componenti Oggetto di Test

Il piano copre la verifica dei seguenti package Java, come definiti nell'Object Design Document:

- **`com.heavyroute.core.controller`:** Componenti responsabili della gestione delle richieste HTTP, della validazione sintattica dei DTO e della conversione delle risposte.
- **`com.heavyroute.core.service`:** Componenti che incapsulano la logica di business, le regole di validazione semantica e l'orchestrazione dei flussi.
- **Interazioni con il Database:** Verifica indiretta tramite test di integrazione che coinvolgono il layer JPA (`Repository`).

3.2. Dipendenze e Mocking

Poiché il test è focalizzato sul Core Business in isolamento, le dipendenze esterne saranno simulate (Mocking):

- **Database:** Sostituito da H2 (In-Memory Database) per i test di integrazione.
- **Servizi Esterni:** Le chiamate a Google Maps API, VIES e al servizio di Notifica saranno sostituite da *Mock Objects* (tramite Mockito) per garantire determinismo e velocità di esecuzione.

4. Features to be tested

Questa sezione identifica le funzionalità di business e tecniche che saranno verificate dai casi di test.

4.1. Features to be Tested

- **Gestione Richieste (UC3):**
  - Creazione corretta di una richiesta di trasporto da parte di un Committente.
  - Validazione rigorosa degli input (es. date, pesi, indirizzi) e gestione degli errori (HTTP 400).
- **Pianificazione Viaggio (UC4):**
  - Logica di approvazione della richiesta e generazione automatica dell'entità `Trip`.
  - Assegnazione delle risorse con verifica della disponibilità.
  - Gestione dei conflitti di risorsa (es. veicolo già impegnato).
- **Validazione Rotta (UC5):**
  - Logica di approvazione della rotta calcolata da parte del Traffic Coordinator.
  - Gestione del rifiuto della rotta e conseguente cancellazione/ripianificazione del viaggio.
- **Sicurezza (NFR Security):**
  - Verifica dei permessi di accesso agli endpoint critici (es. solo il PL può approvare, solo il TC può validare).

4.2. Features Not to be Tested

Le seguenti aree sono esplicitamente escluse da questo piano di test:

- **Interfaccia Utente (Frontend):** La logica di presentazione è testata separatamente.
- **Performance e Carico:** I requisiti non funzionali di performance richiedono un ambiente di produzione e saranno verificati in una fase successiva.
- **Integrazione Reale con Terze Parti:** Non verranno effettuate chiamate reali a servizi a pagamento (es. Google Maps) o governativi (VIES) per evitare costi e dipendenze di rete durante il testing automatico.

5. Pass/Fail criteria

5.1. Criteri per il Singolo Test Case

Un test case è considerato **PASS** solo se soddisfa tutte le seguenti condizioni:

- **Output Funzionale:** Il valore restituito dal metodo o il corpo della risposta HTTP coincide esattamente con l'oracolo definito nel TCS.
- **Stato del Sistema:** Le modifiche al database (side effects) sono corrette e non sono stati alterati dati non pertinenti al test.
- **Gestione Errori:** Nel caso di test negativi, viene sollevata l'eccezione specifica attesa (es. `ResourceNotFoundException`) e restituito il codice di stato HTTP corretto (es. 404).

5.2. Criteri di Accettazione del Sottosistema

L'intera fase di test sarà considerata conclusa con successo se:

- Il **100%** dei test case prioritari ("Critical" e "High") è superato.
- Non sono apparsi difetti di gravità "Bloccante" o "Critica".
- La **Code Coverage** (copertura delle linee di codice) misurata tramite strumento JaCoCo è superiore all'80% per il package `core`.

6. Approach

La strategia di testing combina una metodologia rigorosa per la progettazione dei casi di test con un approccio "bottom-up" per la loro esecuzione.

6.1. Metodologia di Design: Category Partition

Per la specifica dei casi di test funzionali e di sistema, è stata adottata la tecnica del **Category Partition Method (CPM)**. Questo approccio sistematico ha permesso di:

1. Decomporre le specifiche funzionali in unità testabili.
2. Identificare i parametri di input e le condizioni ambientali critiche.
3. Definire categorie e scelte (classi di equivalenza) per ogni parametro, distinguendo tra valori validi, non validi e casi limite (boundary).
4. Generare un set di test case rappresentativi combinando queste scelte, garantendo una copertura efficace dei requisiti funzionali e riducendo la ridondanza.

6.2. Livelli di Esecuzione

L'esecuzione dei test segue una progressione incrementale:

6.2.1. Livello 1: Unit Testing

Obiettivo: Verificare la correttezza della logica di business in isolamento totale.

- **Oggetto:** Le classi del layer `Service`.
- **Tecnica:** White-box testing.
- **Strumenti: JUnit 5** per l'esecuzione, **Mockito** per simulare il comportamento dei Repository e dei servizi esterni.
- **Focus:** Calcoli, transizioni di stato, gestione delle eccezioni di business.

6.2.2. Livello 2: Integration Testing (Component Testing)

Obiettivo: Verificare che i componenti interagiscano correttamente tra loro e con il framework.

- **Oggetto:** L'interazione `Controller` → `Service` → `Repository`.
- **Tecnica:** Black-box testing (test dell'API REST), con input derivati dall'analisi Category Partition.
- **Strumenti: Spring Boot Test** con **MockMvc** per simulare le richieste HTTP e **H2 Database** per la persistenza volatile.
- **Focus:** Mapping corretto degli endpoint, serializzazione JSON, validazione degli input, transazionalità delle operazioni sul DB.

7. Suspension and resumption

7.1. Criteri di Sospensione

Le attività di test saranno immediatamente sospese al verificarsi di una delle seguenti condizioni:

- Identificazione di un difetto "Bloccante" che impedisce l'esecuzione dei test successivi (es. impossibilità di autenticarsi o crash del database di test).
- Instabilità critica dell'ambiente di test (es. configurazione errata che invalida i risultati).

7.2. Criteri di Ripresa

Il testing riprenderà solo quando:

- Il difetto bloccante è stato corretto e verificato con uno "Smoke Test".
- L'ambiente è stato ripristinato a uno stato noto e stabile.

8. Testing materials

Per l'esecuzione dei test pianificati sono richieste le seguenti risorse:

8.1. Hardware

- **Developer Workstations:** PC standard (min. 16GB RAM) per l'esecuzione locale dei test.

8.2. Software Tools

- **Linguaggio:** Java 17 (JDK) o superiore.
- **Framework:** Spring Boot 3.x.
- **Build Tool:** Maven (gestione dipendenze e ciclo di vita del test).
- **Test Frameworks:**
  - **JUnit 5:** Motore di esecuzione dei test.
  - **Mockito:** Libreria per la creazione di mock objects.
  - **Spring Test / MockMvc:** Per i test di integrazione del livello web.
- **Database di Test:** H2 Database (in-memory) per garantire isolamento e velocità.
- **IDE:** IntelliJ IDEA.

9. Test cases

Questa sezione elenca i casi di test pianificati, derivati dall'applicazione del metodo *Category Partition* (per i test funzionali) e dall'analisi architetturale (per i test di sicurezza e unitari). Ogni identificativo (`ID`) fa riferimento a una specifica tecnica dettagliata contenuta nel documento separato *Test Case Specification (TCS)*.

TC-CORE-01 Creazione Richiesta Valida (Unit Test)
Obiettivo: Verificare che il metodo `TransportRequestService.createRequest` persista correttamente una nuova richiesta nel database con lo stato iniziale corretto.

TC-CORE-02.A Validazione Input Richiesta: Peso Negativo (Integration Test)
Obiettivo: Verificare che il `TransportRequestController` intercetti input con peso negativo e restituisca HTTP 400.

TC-CORE-02.B Validazione Input Richiesta: Data Passata (Integration Test)
Obiettivo: Verificare che il sistema rifiuti richieste con data di ritiro nel passato.

TC-CORE-03 Approvazione e Generazione Viaggio (Unit Test)
Obiettivo: Verificare la logica di transizione di stato della richiesta e la contestuale creazione dell'oggetto `Trip` in stato `IN_PLANNING`.

TC-CORE-04 Assegnazione Risorse (Unit Test)
Obiettivo: Verificare che l'associazione di un Autista e un Veicolo a un Viaggio aggiorni lo stato a `WAITING_VALIDATION`.

TC-CORE-05 Gestione Conflitti Risorse (Unit Test)
Obiettivo: Verificare che il sistema impedisca l'assegnazione di un veicolo già impegnato, sollevando un'eccezione di business appropriata.

TC-CORE-06 Sicurezza Endpoint (Integration Test)
Obiettivo: Verificare che i meccanismi di sicurezza (Spring Security) impediscano l'accesso non autorizzato.

TC-CORE-07 Approvazione Rotta (Unit Test)
Obiettivo: Verificare che l'approvazione del Traffic Coordinator porti il viaggio allo stato `CONFIRMED`.

TC-CORE-08 Rifiuto Rotta (Unit Test)
Obiettivo: Verificare che il rifiuto della rotta comporti la cancellazione del viaggio e la notifica al Planner.

10. Testing schedule

Il seguente calendario definisce le milestone per l'esecuzione dei test sul sottosistema Core Business:

- **Fase 1: Preparazione (22 Dicembre)**
  - Setup dell'ambiente di test (configurazione H2, dipendenze Maven).
  - Scrittura degli stub e dei mock per i servizi esterni.
- **Fase 2: Unit Testing (4 Gennaio)**
  - Implementazione ed esecuzione dei test TC-CORE-01, 03, 04, 05, **07, 08**.
  - Verifica della copertura del codice.
- **Fase 3: Integration Testing (6 Gennaio)**
  - Implementazione ed esecuzione dei test **TC-CORE-02.A, 02.B**, 06.
  - Verifica dei flussi HTTP completi.
- **Fase 4: Chiusura (13 Gennaio)**
  - Analisi dei risultati e correzione dei difetti emersi.
  - Generazione del report finale di test.