



# Laurea Triennale in Informatica

Università di Salerno

Corso di Ingegneria del Software

Prof. Andrea De Lucia



## Progetto EasyDrive

### Test Plan

Versione 0.1

**Data:** 21/12/2025

**Coordinatore del progetto:**

<b>Nome</b>	Matricola
<b>Mario Mascheri</b>	0512120157
<b>Claudio Brizio</b>	0512119716

**Partecipanti:**

<b>Nome</b>	Matricola
<b>Brizio Claudio</b>	0512119716
<b>Cannella Vincenzo</b>	0512119065
<b>Coscia Matteo</b>	0512121210
<b>Mascheri Mario</b>	0512120157

**Scritto da:** Vincenzo Cannella

**Cronologia delle Revisioni**

<b>Data</b>	<b>Versione</b>	<b>Descrizione</b>	<b>Autore</b>
21/12/2025	0.1	Prima stesura e bozza del Test Plan	Vincenzo Cannella
04/02/2026	1.0	Stesura ufficiale e definitiva del Test Plan	Vincenzo Cannella

# Indice

<b>1</b>	<b>Introduzione</b>	<b>3</b>
1.1	Obiettivi del Test . . . . .	3
1.2	Estensione del Test (Scope) . . . . .	4
<b>2</b>	<b>Relazione con altri documenti</b>	<b>4</b>
<b>3</b>	<b>Panoramica del sistema</b>	<b>5</b>
3.1	Granularità dei Componenti . . . . .	5
3.2	Dipendenze tra i Componenti . . . . .	6
<b>4</b>	<b>Funzionalità da testare / non testare</b>	<b>6</b>
4.1	Funzionalità da testare . . . . .	6
4.2	Funzionalità non soggette a test . . . . .	7
<b>5</b>	<b>Criteri di Superamento/Fallimento</b>	<b>7</b>
5.1	Criteri di Successo del Test (Identificazione Fallimenti) . . . . .	7
5.2	Criteri di Accettazione del Sistema . . . . .	8
5.3	Gestione degli Incidenti . . . . .	8
<b>6</b>	<b>Approccio</b>	<b>8</b>
6.1	Strategia di Testing Generale . . . . .	9
<b>7</b>	<b>Sospensione e Ripresa</b>	<b>9</b>
7.1	Criteri di Sospensione . . . . .	9
7.2	Criteri di Ripresa . . . . .	10
<b>8</b>	<b>Materiali di Test</b>	<b>10</b>
8.1	Risorse Hardware . . . . .	10
8.2	Risorse Software . . . . .	11

8.3	Strumenti di Testing e Supporto . . . . .	11
8.4	Matrice delle Attività . . . . .	11
<b>9</b>	<b>Pianificazione dei Tempi (Schedule)</b>	<b>12</b>
9.1	Test Cases . . . . .	12
<b>10</b>	<b>Specifiche dei casi di test con tecnica Category Partition</b>	<b>12</b>
10.1	Funzionalità: Registrazione Utente (UC 3) . . . . .	13
10.2	Funzionalità: Login (Autenticazione) . . . . .	14
10.3	Funzionalità: Prenotazione Noleggio . . . . .	15
10.4	Funzionalità: Pagamento e Vendita (UC 15) . . . . .	15
10.5	Funzionalità: Ricerca Veicoli (Filtri Catalogo) . . . . .	16

# 1 Introduzione

L'obiettivo di questa sezione è definire gli scopi e l'ampiezza delle attività di verifica per il sistema Easy Drive. Il presente documento fornisce il framework operativo che i manager e i tester utilizzeranno per pianificare ed eseguire i test necessari in modo tempestivo ed efficiente sotto il profilo dei costi.

## 1.1 Obiettivi del Test

Le attività di testing mirano a garantire che il software soddisfi i requisiti specificati e sia privo di difetti critici prima del rilascio. Gli obiettivi principali includono:

- **Verifica Funzionale:** confermare che ogni sottosistema sia codificato correttamente e svolga le funzionalità previste, come la gestione dei veicoli (RF1), le prenotazioni (RF2), la ricerca avanzata (RF3), la gestione vendite (RF4) e lo storico operazioni (RF5).
- **Verifica dei Requisiti Globali:** determinare se il sistema rispetti i Requisiti Non Funzionali (RNF), con particolare attenzione alle prestazioni (risposta in meno di 2 secondi) e alla disponibilità (99% del tempo).

- **Identificazione dei Fallimenti:** esercitare il software con casi di test specifici per individuare deviazioni tra il comportamento osservato e quello specificato (failure) causate da bug algoritmici o meccanici.
- **Affidabilità e Sicurezza:** validare i meccanismi di protezione dei dati personali tramite HTTPS e cifratura, garantendo l'integrità delle informazioni sensibili.

## 1.2 Estensione del Test (Scope)

Il piano di test copre l'intero sistema Easy Drive, scomposizione inclusa:

- **Test di Componente (Unit Testing):** verifica dei singoli package funzionali, tra cui `it.unisa.easydrive.catalog`, `it.unisa.easydrive.user`, `it.unisa.easydrive.booking`, `it.unisa.easydrive.sales`, `it.unisa.easydrive.payment`, `it.unisa.easydrive.staff` e `it.unisa.easydrive.core`.
- **Test di Integrazione:** verifica delle interfacce e delle interazioni tra i sottosistemi sopracitati per assicurare il corretto flusso di dati.
- **Test di Sistema:** valutazione dell'intero sistema integrato nel suo ambiente target (architettura client-server basata su PHP, Apache e MySQL).
- **Test di Accettazione:** sessioni di test finali condotte per dimostrare che il sistema risponde alle esigenze del cliente e sia pronto per l'uso operativo.

## 2 Relazione con altri documenti

Il presente Piano di Test è strettamente integrato con gli altri documenti prodotti durante il ciclo di vita del software, fungendo da strumento di verifica per le specifiche definite nelle fasi precedenti. Di seguito viene illustrata la relazione tra i test e la documentazione di progetto:

**Requirements Analysis Document (RAD):** il RAD costituisce la base primaria per il Functional Testing. Ogni test case funzionale è progettato per validare i requisiti estratti dagli scenari e dai casi d'uso definiti nel RAD, assicurando che il sistema soddisfi le necessità dell'utente finale.

**System Design Document (SDD):** l'SDD viene utilizzato come riferimento per l'Integration Testing. I test di integrazione verificano che le interazioni tra i sottosistemi (es. Catalogo, Prenotazioni, Pagamenti) avvengano secondo l'architettura client-server e il mapping hardware/software stabiliti.

**Object Design Document (ODD):** l'ODD guida l'esecuzione del Component Testing (Unit Test). I test unitari si concentrano sulla verifica delle interfacce pubbliche delle classi, degli attributi e dei metodi descritti nell'ODD, garantendo che ogni unità di codice sia corretta prima dell'integrazione.

**Problem Statement:** fornisce il contesto di alto livello per i test di accettazione, permettendo di verificare che l'intento originale di automatizzare la gestione dell'agenzia sia stato raggiunto con successo.

## 3 Panoramica del sistema

Questa sezione descrive gli aspetti strutturali del sistema Easy Drive, identificando i componenti che saranno oggetto di test unitari. La strategia di verifica si basa sulla scomposizione del sistema in sottosistemi isolabili, definiti come gruppi di oggetti o package funzionali.

### 3.1 Granularità dei Componenti

Ai fini del Unit Testing, il sistema viene suddiviso in package logici che rappresentano le unità minime di test. Ogni componente è progettato per essere testato individualmente dai programmatori per confermare che la logica interna e le funzionalità previste siano codificate correttamente.

I componenti principali identificati sono i seguenti:

- `it.unisa.easydrive.catalog`: gestisce la navigazione, la ricerca e il filtraggio dei veicoli.
- `it.unisa.easydrive.account`: include i moduli per l'autenticazione, la registrazione e la gestione del profilo utente.
- `it.unisa.easydrive.booking`: implementa la logica di business per il controllo della disponibilità e la gestione dei noleggi.
- `it.unisa.easydrive.sales`: gestisce il flusso d'acquisto, gli ordini e la persistenza dello storico delle transazioni.
- `it.unisa.easydrive.payment`: interfaccia il sistema con i servizi di validazione ed esecuzione dei pagamenti.
- `it.unisa.easydrive.core`: package trasversale contenente le entità condivise (es. Veicolo, Utente) e la logica di connessione al database.

### 3.2 Dipendenze tra i Componenti

La progettazione dei test deve tenere conto delle relazioni gerarchiche e funzionali tra i componenti per determinare l'ordine di esecuzione e l'eventuale necessità di strumenti di supporto (Stubs e Drivers):

- **Dipendenza dai Dati:** tutti i sottosistemi dello strato applicativo (Application Layer) dipendono dal componente dei dati persistenti (Data Layer) per l'accesso al DBMS MySQL.
- **Trasversalità del Core:** il package core rappresenta la base del sistema; le sue entità sono utilizzate da tutti gli altri componenti, rendendo la sua stabilità critica per l'intero processo di testing.
- **Interazione Client-Server:** poiché il sistema adotta un'architettura client-server, il browser (Client Node) dipende dai servizi esposti dal Web Server tramite protocollo HTTP/HTTPS.

## 4 Funzionalità da testare / non testare

Questa sezione identifica i requisiti funzionali e le combinazioni di funzionalità che saranno oggetto di verifica, oltre a specificare quali aspetti del sistema sono esclusi dal piano di test e le relative motivazioni.

### 4.1 Funzionalità da testare

Il focus primario del testing funzionale è validare che il sistema esegua correttamente le operazioni di business descritte nei requisiti. Saranno testate le seguenti funzionalità ,corrispondenti ai sottosistemi definiti nell SDD:

- **Gestione Veicoli (RF1):** inserimento di nuove automobili, modifica delle informazioni (marca, modello, anno, prezzo, disponibilità) ed eliminazione dal catalogo.
- **Gestione Prenotazioni e Noleggi (RF2):** creazione di nuove prenotazioni, registrazione dei noleggi e capacità di modifica o annullamento.
- **Gestione Vendite (RF4):** inserimento delle informazioni relative ai clienti e gestione della procedura di acquisto.

- **Combinazioni di Funzionalità:** test integrati su flussi complessi, come la ricerca di un veicolo seguita dalla sua prenotazione e dal relativo pagamento.

## 4.2 Funzionalità non soggette a test

Le seguenti caratteristiche non saranno verificate internamente dal team di sviluppo per le ragioni specificate:

- **Gestione dell'Infrastruttura Server:** operazioni di avvio (start-up), terminazione (shutdown) e configurazione hardware del server.  
*Motivazione:* queste attività sono gestite automaticamente dall'ambiente web predefinito (Web Server Apache/Nginx e runtime PHP) e non appartengono alla logica applicativa del software.
- **Logica Interna del Sistema di Pagamento Esterno:** elaborazione effettiva del credito bancario al di fuori dell'invio dei dati e della ricezione dell'esito.  
*Motivazione:* il sistema si interfaccia con servizi esterni; viene testata solo l'interfaccia di comunicazione e la gestione dei risultati (successo, fallimento, timeout).
- **Compatibilità con Browser Obsoleti:** browser che non supportano i moderni standard HTML, CSS e JavaScript.  
*Motivazione:* il sistema è progettato per supportare specificamente le versioni correnti di Chrome, Edge, Firefox e Safari.

## 5 Criteri di Superamento/Fallimento

In questa sezione vengono stabiliti i criteri generici per determinare l'esito dei test. Un test è considerato "di successo" quando riesce a causare un fallimento (failure), rivelando la presenza di un bug. Al contrario, un test che non rileva anomalie conferma la conformità del componente alla specifica, ma non ne garantisce l'assenza totale di difetti.

Un test ha successo (pass) se, dato un input al sistema, l'output ottenuto è diverso dall'output atteso dall'oracolo. Un test fallisce (fail) se, dato un input al sistema, l'output ottenuto è uguale all'output atteso dall'oracolo.

### 5.1 Criteri di Successo del Test (Identificazione Fallimenti)

Un test individuale è considerato riuscito se si verifica una delle seguenti condizioni:

- **Deviazione dalla Specifica:** il comportamento osservato del sistema differisce dal comportamento specificato nei requisiti funzionali (RF1-RF5).
- **Stato Erroneo:** il sistema entra in uno stato tale per cui l'elaborazione successiva porterà inevitabilmente a un fallimento.

## 5.2 Criteri di Accettazione del Sistema

Affinché il sistema o un sottosistema superi la fase di verifica e sia considerato pronto per l'uso, devono essere soddisfatti i seguenti parametri qualitativi:

### A. Requisiti Funzionali

RF1-RF5: tutte le operazioni principali (gestione veicoli, prenotazioni, ricerca, vendite e storico) devono operare senza errori algoritmici o meccanici.

Robustezza: il sistema deve gestire correttamente gli input errati o non validi, notificando l'utente senza compromettere l'integrità dei dati.

## 5.3 Gestione degli Incidenti

Ogni volta che un test risulta "di successo" (ovvero trova un bug), viene generato un Test Incident Report. Questo documento descrive l'incidente per permettere agli sviluppatori di:

- analizzare e dare priorità al fallimento;
- pianificare le modifiche necessarie al sistema;
- eseguire i test di regressione per assicurarsi che la correzione non abbia introdotto nuovi difetti.

## 6 Approccio

Questa sezione descrive la metodologia generale adottata per il processo di verifica del sistema Easy Drive, dettagliando la strategia di integrazione selezionata e le tecniche di supporto utilizzate.

## 6.1 Strategia di Testing Generale

Il processo segue un modello iterativo in cui l'implementazione del codice e il testing procedono di pari passo. Le attività sono suddivise in quattro fasi principali condotte dagli sviluppatori e, nella fase finale, dal cliente:

- Unit Testing: verifica dei singoli componenti isolati.
- Integration Testing: test delle interfacce tra i sottosistemi.
- System Testing: verifica dell'intero sistema rispetto ai requisiti funzionali e globali.
- Acceptance Testing: valutazione finale condotta dal cliente per dimostrare che il sistema è pronto all'uso.

## 7 Sospensione e Ripresa

Questa sezione specifica i criteri necessari per sospendere temporaneamente le attività di test e le condizioni indispensabili per la loro ripresa, al fine di ottimizzare le risorse ed evitare test inutili su componenti instabili.

### 7.1 Criteri di Sospensione

Le attività di testing verranno immediatamente sospese se si verifica una delle seguenti condizioni critiche:

- **Bug Bloccanti**: identificazione di un difetto che impedisce l'esecuzione di un intero flusso di lavoro fondamentale (es. impossibilità di accedere al sistema o crash del database MySQL durante la ricerca dei veicoli).
- **Instabilità dell'Ambiente**: malfunzionamento critico dell'infrastruttura server (Apache o interprete PHP) che impedisce la comunicazione tra Client e Server.
- **Violazione della Sicurezza**: rilevamento di vulnerabilità che espongono i dati sensibili degli utenti in chiaro o malfunzionamento dei certificati HTTPS.
- **Mancanza di Documentazione Corretta**: se viene rilevato che il codice non corrisponde alle specifiche dell'ODD o dell'SDD, rendendo i test non attendibili.

## 7.2 Criteri di Ripresa

Le attività di test potranno essere riprese solo dopo che i criteri di sospensione saranno stati risolti e validati. Le condizioni per la ripresa includono:

- **Correzione dei Difetti (Correction):** gli sviluppatori devono aver applicato le modifiche necessarie per riparare il guasto che ha causato il fallimento.
- **Ripristino dell'Operatività:** in caso di errore hardware o software critico, l'operatività deve essere ripristinata entro un tempo massimo di 10 minuti, come specificato nei requisiti di affidabilità.
- **Esecuzione dei Test di Regressione:** prima di procedere con nuovi test, devono essere ri-eseguiti i test precedenti per assicurarsi che le correzioni non abbiano introdotto nuovi bug (regression testing).
- **Verifica dell'Integrità dei Dati:** conferma che il database sia stato ripristinato tramite backup giornalieri se la sospensione è stata causata da una perdita di dati.

# 8 Materiali di Test

In questa sezione vengono identificate le risorse hardware, software e gli strumenti di supporto necessari per allestire l'ambiente di prova e rendere eseguibili i casi di test definiti.

## 8.1 Risorse Hardware

Per garantire la validità dei test di sistema e di accettazione, l'hardware deve rispecchiare l'ambiente target descritto nel Problem Statement:

- **Lato Client (Postazioni Tester):**
  - Computer desktop o laptop con almeno 8 GB di RAM.
  - Dispositivi mobili (Smartphone/Tablet) con sistemi operativi Android o iOS per testare la responsività della piattaforma.
  - Connessione internet stabile per le prove di latenza e performance.
- **Lato Server (Ambiente di Staging):**
  - Server dedicato o istanza virtuale configurata per ospitare l'applicazione e il database.

## 8.2 Risorse Software

L'ambiente di test deve replicare fedelmente lo stack tecnologico scelto per lo sviluppo:

- **Sistemi Operativi:** Windows 10 o superiore, Linux (Ubuntu/CentOS), macOS.
- **Web Server:** Apache o Nginx.
- **Linguaggio di Scripting:** PHP (versione compatibile con il codice sorgente).
- **Database Management System:** MySQL per la gestione della persistenza dei dati.
- **Browser Web:** versioni aggiornate di Chrome, Firefox, Edge e Safari (per la verifica del requisito RNF6 - Portabilità).

## 8.3 Strumenti di Testing e Supporto

Per l'automatizzazione e la gestione dei report, verranno utilizzati i seguenti tool:

- **PHPUnit:** framework di riferimento per l'esecuzione degli Unit Test e per il calcolo della copertura del codice (Code Coverage).
- **Selenium :** per l'automazione dei test funzionali sulla UI (User Interface) e per i test di sistema.
- **Git / GitHub:** per il controllo di versione degli script di test e per garantire la tracciabilità delle modifiche.

## 8.4 Matrice delle Attività

Di seguito sono elencate le attività di testing e i relativi output richiesti:

- **Pianificazione**

*Responsabile:* Coordinatore Progetto

*Output:* Piano di Test (questo documento)

- **Progettazione Test**

*Responsabile:* Analista / Progettista

*Output:* Test Case Specification

- **Esecuzione Test**

*Responsabile:* Tester / Sviluppatori

*Output:* Test Log / Test Execution Report

- **Segnalazione Bug**

*Responsabile:* Tester

*Output:* Test Incident Report

## 9 Pianificazione dei Tempi (Schedule)

Le attività di pianificazione del testing saranno avviate al termine della fase di progettazione, in linea con quanto descritto nei capitoli precedenti.

La definizione e la redazione dei casi di test procederanno in parallelo alle attività di sviluppo del software.

L'esecuzione delle attività di test avrà luogo a seguito del completamento dell'implementazione del sistema. Al termine dello sviluppo, l'intera suite di test verrà rieseguita al fine di verificare il corretto funzionamento dell'applicazione e predisporre la reportistica finale.

### 9.1 Test Cases

Per lo sviluppo dei test cases verrà utilizzato il metodo del *Category Partition*. Questo metodo consiste nell'identificare, per ciascuna funzionalità da testare, un insieme di parametri. Per ogni parametro vengono definite delle categorie, che vengono successivamente suddivise in scelte. A ciascuna scelta viene infine assegnato un valore.

I test cases verranno definiti nel paragrafo *Test Cases Specification*.

## 10 Specifica dei casi di test con tecnica Category Partition

In questa sezione viene definita la strategia di verifica funzionale del sistema **EasyDrive**. La tecnica della *Category Partition* viene utilizzata per partizionare il dominio di input di ogni funzionalità in classi di equivalenza, identificando le scelte rappresentative e i relativi vincoli per generare un set di test esaustivo ma non ridondante.

## 10.1 Funzionalità: Registrazione Utente (UC 3)

**Obiettivo:** Verificare la corretta creazione di un account cliente con validazione anagrafica, residenza e sicurezza dei dati.

**Parametri:** Username, Email, Password, Nome, Cognome, Data\_Nascita, Telefono, Indirizzo, CAP, Città, Provincia.

**Oggetti dell'ambiente:** Database Utenti.

### Categorie e Scelte

- **Categoria 1: Formato Email**

- EF1: Formato email valido (es. nome@dominio.it) [property email\_corretta]
- EF2: Formato email NON corretto [Error]

- **Categoria 2: Unicità Credenziali (Email e Username)**

- UN1: Email e Username NON presenti nel database [if email\_corretta] [property credenziali\_univoche]
- UN2: Email già esistente nel sistema [Error] [if email\_corretta]
- UN3: Username già occupato da un altro profilo [Error] [if email\_corretta]

- **Categoria 3: Robustezza Password**

- RP1: Almeno 8 caratteri, una maiuscola e un numero [property password\_robusta]
- RP2: Password troppo breve o priva di numeri/maiuscole [Error]

- **Categoria 4: Requisito di Età (Data di Nascita)**

- AG1: Utente maggiorenne ( $\geq 18$  anni alla data odierna) [if credenziali\_univoche] [property maggiorenne]
- AG2: Utente minorenne [Error] [if credenziali\_univoche]

- **Categoria 5: Validità Formato CAP**

- CAP1: CAP numerico di esattamente 5 cifre [if maggiorenne]
- CAP2: CAP non numerico o con lunghezza errata [Error] [if maggiorenne]

## Test Frames - Registrazione Utente

- **TF\_REG\_01:** EF1, UN1, RP1, AG1, CAP1 → **Esito Atteso:** Successo (Account creato).
- **TF\_REG\_02:** EF2, -, -, -, - → **Esito Atteso:** Errore (Formato Email non valido).
- **TF\_REG\_03:** EF1, UN2, -, -, - → **Esito Atteso:** Errore (Email già presente).
- **TF\_REG\_04:** EF1, UN1, RP2, -, - → **Esito Atteso:** Errore (Password non sicura).
- **TF\_REG\_05:** EF1, UN1, RP1, AG2, - → **Esito Atteso:** Errore (Utente minorenne).
- **TF\_REG\_06:** EF1, UN1, RP1, AG1, CAP2 → **Esito Atteso:** Errore (CAP non valido).

## 10.2 Funzionalità: Login (Autenticazione)

**Parametri:** identificativo (email/username), password.

**Oggetti dell'ambiente:** Database Utenti.

### Categorie e Scelte

- **Categoria 1: Esistenza account nel database**
  - ACC1: Username o email presente nel database [property account\_esiste]
  - ACC2: USername o email non presente nel database [Error]
- **Categoria 2: Validità Password associata**
  - PV1: Password corretta per l'account specificato [if account\_esiste]
  - PV2: Password non corretta [Error] [if account\_esiste]

**Test Frames - Login** Di seguito sono riportate le combinazioni delle scelte (frame) identificate per la funzionalità di Autenticazione:

- **TF\_LGN\_01:** ACC1, PV1 → **Esito Atteso:** Utente autenticato e indirizzato alla sua home page.
- **TF\_LGN\_02:** ACC1, PV2 → **Esito Atteso:** Visualizzazione pagina di autenticazione con messaggio di errore “username o password non corretta”.
- **TF\_LGN\_03:** ACC2, - → **Esito Atteso:** Visualizzazione pagina di autenticazione con messaggio di errore “username o password non corretta”.

## 10.3 Funzionalità: Prenotazione Noleggio

**Parametri:** data\_inizio, data\_fine, veicolo.

**Oggetti dell'ambiente:** Database Veicoli, Calendario Prenotazioni.

### Categorie e Scelte

- **Categoria 1: Coerenza temporale**

- ST1: Data Inizio < Data Fine [property date\_valide]
- ST2: Data Inizio  $\geq$  Data Fine [Error]

- **Categoria 2: Stato di disponibilità del veicolo**

- DV1: Veicolo disponibile nelle date richieste [if date\_valide]
- DV2: Veicolo occupato (già prenotato) [Error] [if date\_valide]

**Test Frames** Di seguito vengono riportate le combinazioni delle scelte identificate (frame):

- **TF\_NOL\_01:** ST1, DV1  $\rightarrow$  **Esito Atteso:** Successo (Prenotazione effettuata)
- **TF\_NOL\_02:** ST1, DV2  $\rightarrow$  **Esito Atteso:** Errore (Veicolo non disponibile)
- **TF\_NOL\_03:** ST2, –  $\rightarrow$  **Esito Atteso:** Errore (Cronologia date non valida)

## 10.4 Funzionalità: Pagamento e Vendita (UC 15)

**Obiettivo:** Verificare la validità della transazione economica attraverso il controllo sequenziale dei dati della carta.

**Parametri:** numero\_carta, scadenza (MM/AA), cvv.

**Oggetti dell'ambiente:** Circuito di pagamento (Mocked).

### Categorie e Scelte

- **Categoria 1: Formato Numero Carta**

- NC1: Numero di 16 cifre numeriche [property numero\_ok]
- NC2: Formato non valido (meno di 16 cifre o caratteri non numerici) [Error]

- **Categoria 2: Validità Temporale (Scadenza)**
  - EX1: Data di scadenza futura rispetto alla data odierna [if numero\_ok] [property scadenza\_ok]
  - EX2: Carta scaduta (data nel passato) [Error] [if numero\_ok]
- **Categoria 3: Formato CVV**
  - CV1: Codice di 3 cifre numeriche [if scadenza\_ok]
  - CV2: Formato non valido (meno di 3 cifre o caratteri non numerici) [Error] [if scadenza\_ok]

#### **Test Frames - Pagamento e Vendita**

- **TF\_PAG\_01:** NC1, EX1, CV1 → **Esito Atteso:** Successo (Pagamento autorizzato)
- **TF\_PAG\_02:** NC2, –, – → **Esito Atteso:** Errore (Numero carta non valido)
- **TF\_PAG\_03:** NC1, EX2, – → **Esito Atteso:** Errore (Carta di credito scaduta)
- **TF\_PAG\_04:** NC1, EX1, CV2 → **Esito Atteso:** Errore (Formato CVV non valido)

## **10.5 Funzionalità: Ricerca Veicoli (Filtri Catalogo)**

**Obiettivo:** Verificare il corretto funzionamento dei filtri di ricerca con selezione gerarchica (Categoria → Modello) e range di prezzo.

**Parametri:** categoria, modello, prezzo\_min, prezzo\_max, marca.

#### **Categorie e Scelte**

- **Categoria 1: Validità Categoria**
  - CE1: Categoria esistente nel sistema (es. SUV) [property cat\_ok]
  - CE2: Categoria non esistente o non valida [Error]
- **Categoria 2: Validità Modello**
  - MO1: Modello esistente e appartenente alla categoria [if cat\_ok] [property mod\_ok]
  - MO2: Modello non esistente o non appartenente [Error] [if cat\_ok]
- **Categoria 3: Consistenza Range di Prezzo**

- PR1:  $prezzo\_min \leq prezzo\_max$  [if mod\_ok] [property range\_ok]
- PR2:  $prezzo\_min > prezzo\_max$  [Error] [if mod\_ok]

### Test Frames - Ricerca Catalogo

- **TF\_CAT\_01:** CE1, MO1, PR1 → **Esito Atteso:** Visualizzazione dei veicoli filtrati.
- **TF\_CAT\_02:** CE2, –, – → **Esito Atteso:** Errore (Categoria non valida).
- **TF\_CAT\_03:** CE1, MO2, – → **Esito Atteso:** Errore (Modello non trovato).
- **TF\_CAT\_04:** CE1, MO1, PR2 → **Esito Atteso:** Errore (Il prezzo minimo supera il massimo).