

AutoFlow
Test Execution Report ed Incident Report
Versione 1.0



Data: 08/11/2025

Sommario

Nessuna voce di sommario trovata.

1. Introduzione

Questo documento presenta i risultati di una sessione di test condotta sul sistema AutoFlow, con l'obiettivo di verificare che il comportamento dell'applicazione sia conforme alle funzionalità richieste nelle fasi di analisi e progettazione. La suite eseguita deriva dai casi di test descritti nella TCS (Test Case Specification) ed è stata applicata alla versione attuale del backend e alle principali funzionalità del flusso utente. Il Test Execution Report raccoglie gli esiti dei casi eseguiti, le condizioni di test, gli eventuali problemi rilevati e una sintesi della copertura del codice ottenuta. L'intento è fornire una visione chiara e strutturata dello stato di qualità del software, utile sia per il team di sviluppo che per la pianificazione delle fasi successive.

2. Relazione con gli altri documenti

Il presente progetto prende forma a partire dall'analisi e dal confronto con soluzioni digitali già affermate nel settore della gestione dei processi commerciali e configurativi in ambito automotive, che rappresentano un riferimento consolidato in termini di efficienza e affidabilità.

Di seguito si presenta un elenco dei documenti chiave del progetto a cui si fa esplicito riferimento:

- Test Plan (TP): documento di pianificazione della fase di test del sistema.
- Test Case Specification (TCS): documento che descrive in dettaglio una serie di test case, specificando input, azioni, condizioni di esecuzione e risultati attesi.

Oltre ai documenti del progetto, si fa riferimento ad opere di letteratura tecnica che hanno contribuito allo sviluppo metodologico e concettuale di questo lavoro:

- Object-Oriented Software Engineering Using UML, Patterns, and Java™ Third Edition di Bernd Bruegge & Allen H. Dutoit.

3. Test Item Transmittal Report

Per l'esecuzione dei test previsti nel progetto AutoFlow è stata utilizzata la suite di classi presenti nel package `src/test/java`, organizzata secondo quanto definito nella TCS. I test unitari sono stati implementati impiegando oggetti mock per isolare le dipendenze e verificare il comportamento dei singoli servizi.

Per quanto riguarda i test di integrazione, sia quelli che coinvolgono il livello di persistenza sia quelli relativi ai controller, è stato utilizzato un database H2 in-memory, configurato appositamente per l'ambiente di test. Ciò ha permesso di eseguire le verifiche senza la necessità di avviare un database esterno, garantendo rapidità e ripetibilità delle esecuzioni. Gli integration controller sono stati testati tramite MockMvc, simulando richieste HTTP verso gli endpoint del backend. La suite può essere eseguita interamente tramite un IDE come IntelliJ, oppure attraverso il comando `mvn test`, senza richiedere componenti aggiuntivi esterni.

Per i test di sistema, è stato utilizzato Selenium IDE in esecuzione su browser Mozilla Firefox, dopo aver importato gli script nello specifico progetto di test. Alcuni scenari richiedono l'interazione manuale con l'interfaccia, in particolare nelle operazioni che prevedono variazioni delle credenziali di accesso, ma la maggior parte dei flussi risulta automatizzabile.

(Per i casi di test TC4 e TC5 sono stati eseguiti integralmente i test di unità e di integrazione; tuttavia, non è stato possibile automatizzare tramite Selenium alcuni scenari specifici, poiché l'interfaccia del sistema impedisce intenzionalmente l'inserimento di input non validi o situazioni non previste dai flussi standard. Tali limitazioni lato front-end rendono impossibile riprodurre artificialmente certe condizioni di errore, già prevenute dal design stesso dell'applicazione.)

4. Test Log

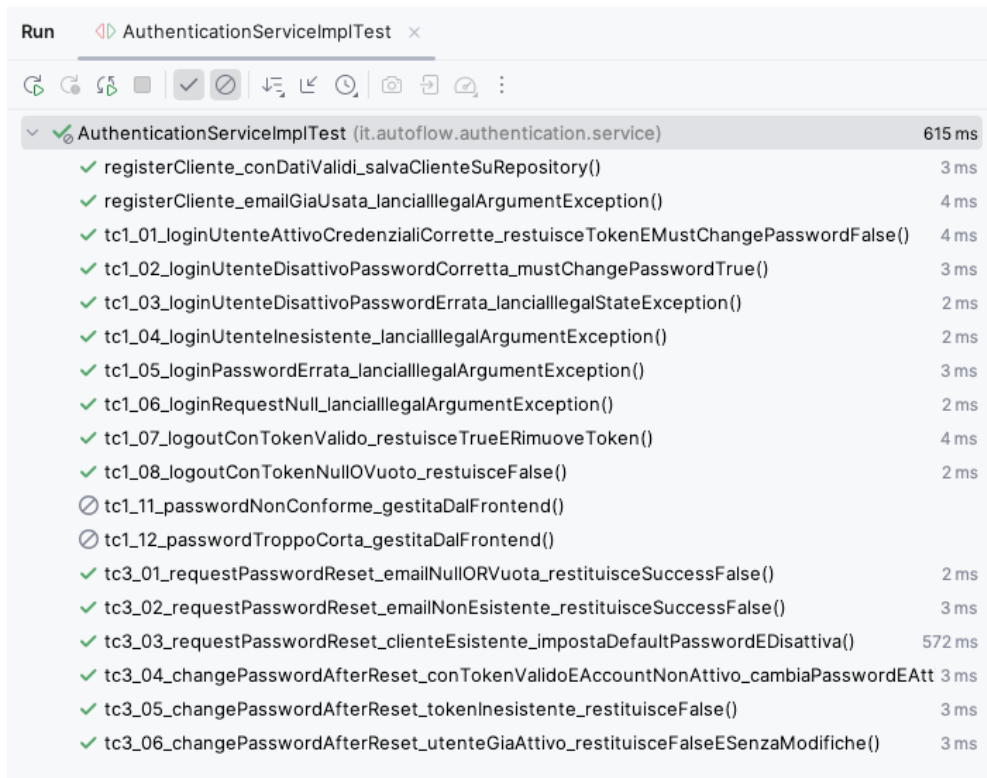
Di seguito sono riportati i risultati dell'esecuzione dei casi di test definiti nella Test Case Specification. Per ciascun caso viene indicato l'esito della prova e una sintesi delle tipologie di test utilizzate per verificarne il corretto comportamento. Le attività di testing sono state svolte impiegando JUnit 5 per l'organizzazione e l'esecuzione dei test, Mockito per la creazione di mock e la validazione dei servizi isolati, e il database H2 in-memory per i test di integrazione che coinvolgono il livello di persistenza. Gli integration controller sono stati verificati mediante MockMvc, simulando richieste HTTP verso gli endpoint dell'applicazione. Per la validazione dei flussi utente è stato inoltre utilizzato Selenium IDE, in esecuzione su Mozilla Firefox, al fine di riprodurre scenari di navigazione reali.

4.1 TC1 - Autenticazione

Obiettivo: verificare il caso d'uso di login utente (cliente/staff), controllando la validità delle credenziali, l'attivazione dell'account e la gestione dei messaggi di errore.

Unit test

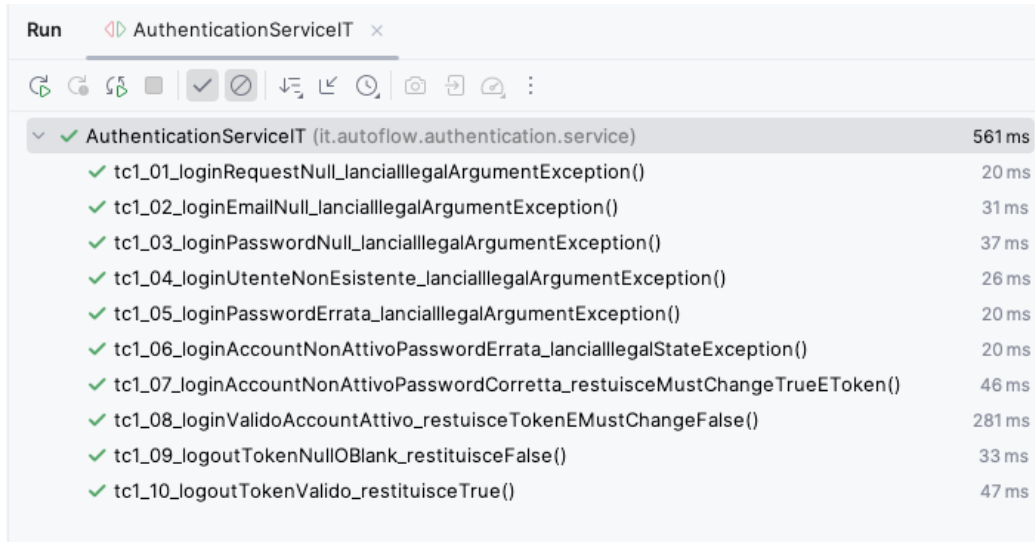
I test unitari hanno riguardato i metodi di *AuthenticationService* responsabili della validazione di email e password. Sono stati utilizzati Mockito e JUnit 5 per simulare il comportamento del repository utenti e dell'encoder delle password. Tutti i test di unità relativi a TC1 sono risultati **Passati**.



Test Case	Duration
AuthenticationServiceImplTest (it.autoflow.authentication.service)	615 ms
✓ registerCliente_conDatiValidi_salvaClienteSuRepository()	3 ms
✓ registerCliente_emailGiaUsata_lanciaIllegalArgument()	4 ms
✓ tc1_01_loginUtenteAttivoCredenzialiCorrette_restituisceTokenEMustChangePasswordFalse()	4 ms
✓ tc1_02_loginUtenteDisattivoPasswordCorretta_mustChangePasswordTrue()	3 ms
✓ tc1_03_loginUtenteDisattivoPasswordErrata_lanciaIllegalStateException()	2 ms
✓ tc1_04_loginUtenteInesistente_lanciaIllegalArgument()	2 ms
✓ tc1_05_loginPasswordErrata_lanciaIllegalArgument()	3 ms
✓ tc1_06_loginRequestNull_lanciaIllegalArgument()	2 ms
✓ tc1_07_logoutConTokenValido_restituisceTrueERimuoveToken()	4 ms
✓ tc1_08_logoutConTokenNullOVuoto_restituisceFalse()	2 ms
✗ tc1_11_passwordNonConforme_gestitaDalFrontend()	
✗ tc1_12_passwordTroppaCorta_gestitaDalFrontend()	
✓ tc3_01_requestPasswordReset_emailNullORVuota_restituisceSuccessFalse()	2 ms
✓ tc3_02_requestPasswordReset_emailNonEsistente_restituisceSuccessFalse()	3 ms
✓ tc3_03_requestPasswordReset_clienteEsistente_impostaDefaultPasswordEDisattiva()	572 ms
✓ tc3_04_changePasswordAfterReset_conTokenValidoEAccountNonAttivo_cambiaPasswordEAtt	3 ms
✓ tc3_05_changePasswordAfterReset_tokenInesistente_restituisceFalse()	3 ms
✓ tc3_06_changePasswordAfterReset_utenteGiaAttivo_restituisceFalseESenzaModifiche()	3 ms

Integration service con repository (H2)

In un ambiente Spring configurato con database **H2 in-memory**, sono stati verificati gli scenari previsti: email inesistente, password errata, account disattivato e accesso valido. Tutti i test relativi ai frame TC1_01–TC1_12 risultano **Passati**.



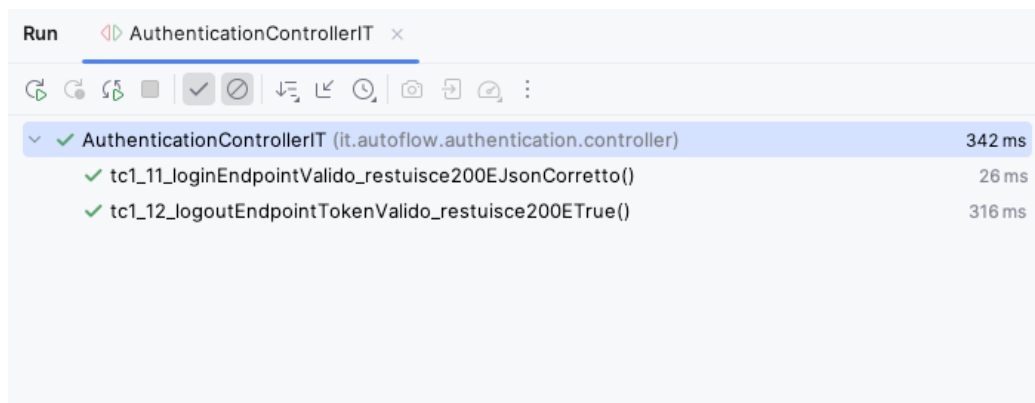
The screenshot shows a JUnit test runner interface with the title 'Run AuthenticationServiceIT'. Below the title is a toolbar with various icons. The main area displays a list of test cases, all of which are marked with a green checkmark, indicating they passed. The test cases are grouped under the package 'it.autoflow.authentication.service'. The total execution time for the suite is 561 ms.

Test Case	Duration
AuthenticationServiceIT (it.autoflow.authentication.service)	561 ms
tc1_01_loginRequestNull_lanciaIllegalArgumentException()	20 ms
tc1_02_loginEmailNull_lanciaIllegalArgumentException()	31 ms
tc1_03_loginPasswordNull_lanciaIllegalArgumentException()	37 ms
tc1_04_loginUtenteNonEsistente_lanciaIllegalArgumentException()	26 ms
tc1_05_loginPasswordErrata_lanciaIllegalArgumentException()	20 ms
tc1_06_loginAccountNonAttivoPasswordErrata_lanciaIllegalStateException()	20 ms
tc1_07_loginAccountNonAttivoPasswordCorretta_restuisceMustChangeTrueEToken()	46 ms
tc1_08_loginValidoAccountAttivo_restuisceTokenEMustChangeFalse()	281 ms
tc1_09_logoutTokenNullOBlank_restituisceFalse()	33 ms
tc1_10_logoutTokenValido_restituisceTrue()	47 ms

Integration controller

Gli endpoint di login sono stati testati mediante MockMvc, verificando, solo **TC1_11** e **TC1_12** perché *tutti gli altri casi del TC1* (errori su email, password, utente non trovato, account non attivo, ecc.) riguardano esclusivamente **la logica del service**, non del controller.

Il controller fa solo da *gateway* HTTP e non contiene condizioni o validazioni proprie: delega tutto all'**AuthenticationService**.

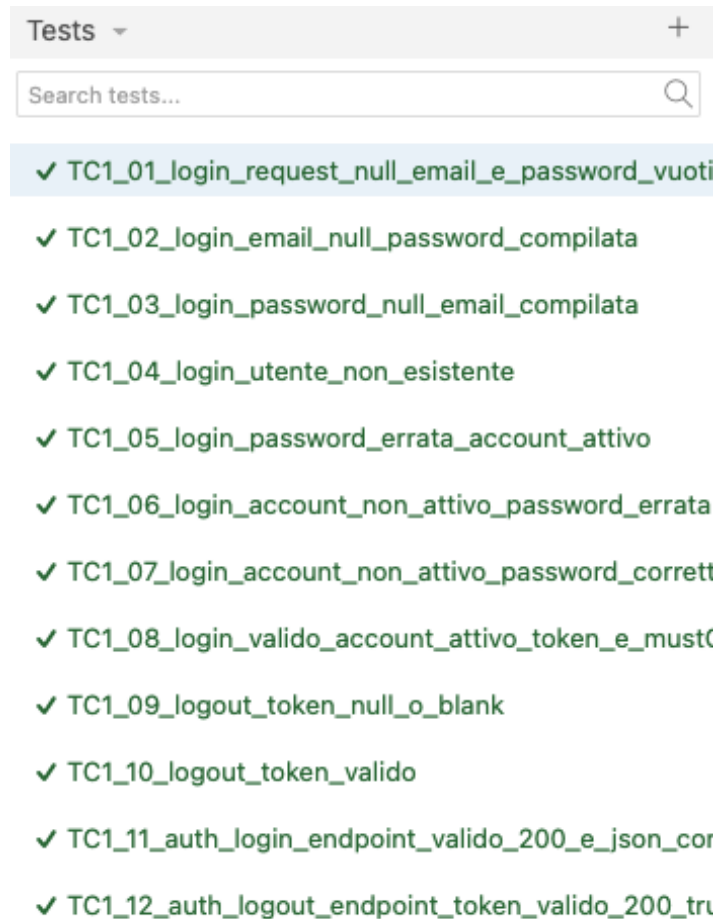


The screenshot shows a JUnit test runner interface with the title 'Run AuthenticationControllerIT'. Below the title is a toolbar with various icons. The main area displays a list of test cases, all of which are marked with a green checkmark, indicating they passed. The test cases are grouped under the package 'it.autoflow.authentication.controller'. The total execution time for the suite is 342 ms.

Test Case	Duration
AuthenticationControllerIT (it.autoflow.authentication.controller)	342 ms
tc1_11_loginEndpointValido_restuisce200EJsonCorretto()	26 ms
tc1_12_logoutEndpointTokenValido_restuisce200ETrue()	316 ms

Test di sistema

Con Selenium IDE su Firefox sono stati simulati i flussi di login lato cliente e lato staff, ricreando scenari corretti e non. Sono stati verificati: validazioni client-side, messaggi di errore ed eventuale reindirizzamento verso la dashboard. L'intera suite di TC1 risulta **Passata**.



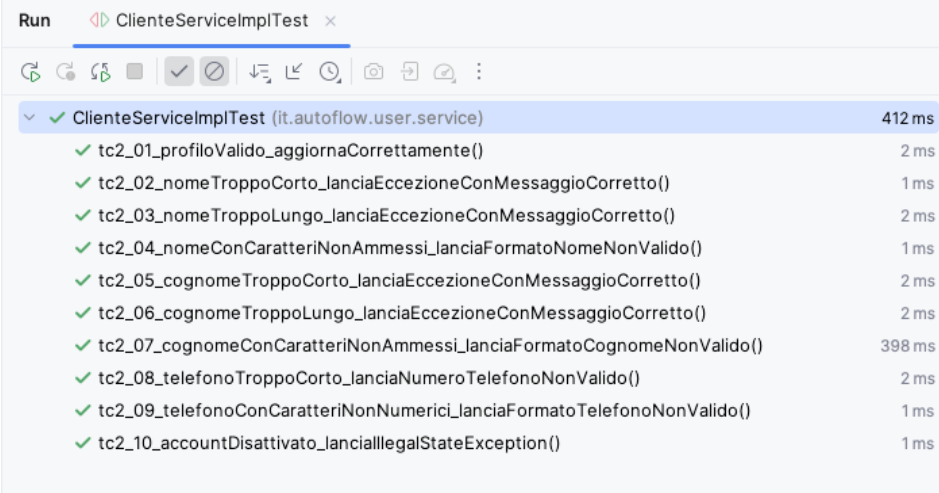
4.2 TC2-TC3 - Modifica profilo cliente e cambio password

TC2: aggiornamento di nome, cognome e telefono con validazione dei formati.

TC3: gestione del cambio password con verifica della password attuale e dei nuovi criteri di sicurezza.

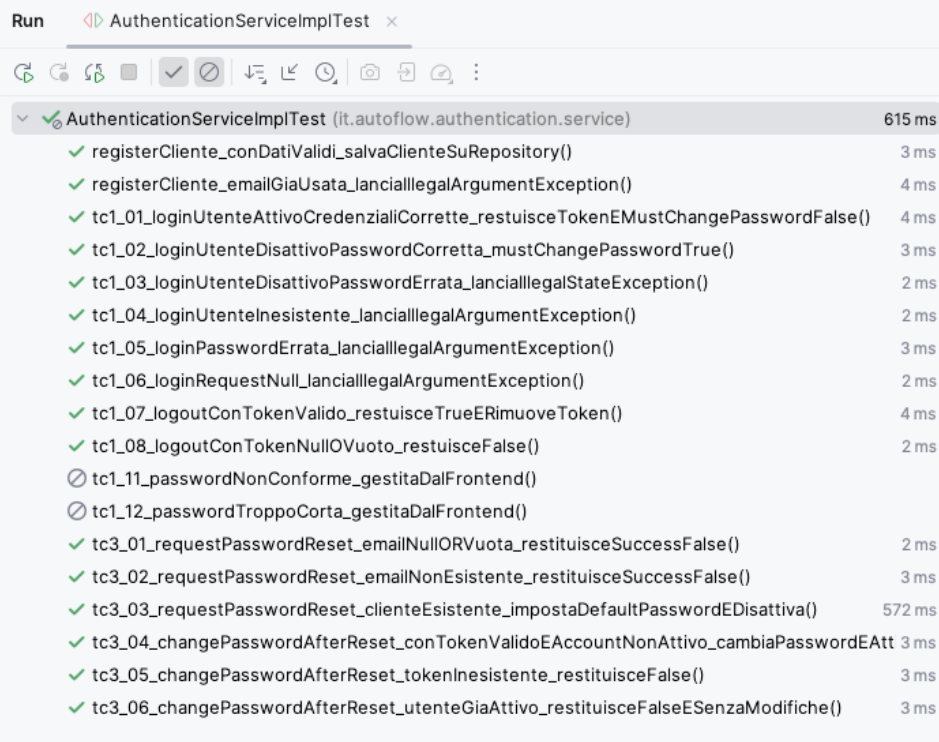
Unit test

I test unitari implementati in *ClienteService* hanno verificato le regole di validazione dei campi anagrafici e della password. Con Mockito è stato possibile isolare la logica del servizio. Tutti i test relativi a TC2 e TC3 sono **Passati**.



Run ClienteServiceImplTest x

✓ ClienteServiceImplTest (it.autoflow.user.service)	412 ms
✓ tc2_01_profiloValido_aggiornaCorrettamente()	2 ms
✓ tc2_02_nomeTroppoCorto_lanciaEccezioneConMessaggioCorretto()	1 ms
✓ tc2_03_nomeTroppoLungo_lanciaEccezioneConMessaggioCorretto()	2 ms
✓ tc2_04_nomeConCaratteriNonAmmessi_lanciaFormatoNomeNonValido()	1 ms
✓ tc2_05_cognomeTroppoCorto_lanciaEccezioneConMessaggioCorretto()	2 ms
✓ tc2_06_cognomeTroppoLungo_lanciaEccezioneConMessaggioCorretto()	2 ms
✓ tc2_07_cognomeConCaratteriNonAmmessi_lanciaFormatoCognomeNonValido()	398 ms
✓ tc2_08_telefonoTroppoCorto_lanciaNumeroTelefonoNonValido()	2 ms
✓ tc2_09_telefonoConCaratteriNonNumerici_lanciaFormatoTelefonoNonValido()	1 ms
✓ tc2_10_accountDisattivato_lanciaIllegalStateException()	1 ms

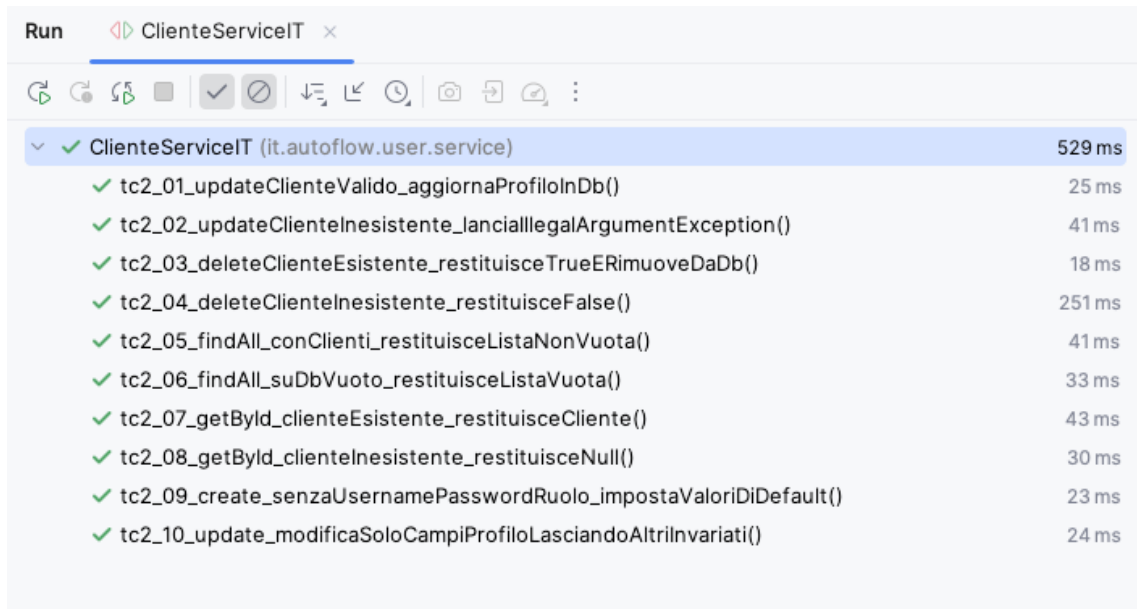


Run AuthenticationServiceImplTest x

✓ AuthenticationServiceImplTest (it.autoflow.authentication.service)	615 ms
✓ registerCliente_conDatiValidi_salvaClienteSuRepository()	3 ms
✓ registerCliente_emailGiaUsata_lanciaIllegalArgumentOutOfRangeException()	4 ms
✓ tc1_01_loginUtenteAttivoCredenzialiCorrette_restuisceTokenEMustChangePasswordFalse()	4 ms
✓ tc1_02_loginUtenteDisattivoPasswordCorretta_mustChangePasswordTrue()	3 ms
✓ tc1_03_loginUtenteDisattivoPasswordErrata_lanciaIllegalStateException()	2 ms
✓ tc1_04_loginUtenteInesistente_lanciaIllegalArgumentOutOfRangeException()	2 ms
✓ tc1_05_loginPasswordErrata_lanciaIllegalArgumentOutOfRangeException()	3 ms
✓ tc1_06_loginRequestNull_lanciaIllegalArgumentOutOfRangeException()	2 ms
✓ tc1_07_logoutConTokenValido_restuisceTrueERimuoveToken()	4 ms
✓ tc1_08_logoutConTokenNullOVuoto_restuisceFalse()	2 ms
✗ tc1_11_passwordNonConforme_gestitaDalFrontend()	
✗ tc1_12_passwordTroppoCorta_gestitaDalFrontend()	
✓ tc3_01_requestPasswordReset_emailNullORVuota_restituisceSuccessFalse()	2 ms
✓ tc3_02_requestPasswordReset_emailNonEsistente_restituisceSuccessFalse()	3 ms
✓ tc3_03_requestPasswordReset_clienteEsistente_impostaDefaultPasswordEDisattiva()	572 ms
✓ tc3_04_changePasswordAfterReset_conTokenValidoEAccountNonAttivo_cambiaPasswordEAttiva()	3 ms
✓ tc3_05_changePasswordAfterReset_tokenInesistente_restituisceFalse()	3 ms
✓ tc3_06_changePasswordAfterReset_utenteGiaAttivo_restituisceFalseESenzaModifiche()	3 ms

Integration service con repository (H2)

Nelle attività di integrazione è stato verificato che il sistema persista correttamente i dati validi e sollevi le eccezioni previste in caso di input non conforme alle regole di business. È stata inoltre confermata la corretta gestione della sicurezza, con il salvataggio delle password esclusivamente in forma hashata. Tutti i test da TC2_01 a TC2_10 e da TC3_01 a TC3_06 sono risultati superati, attestando la solidità dell'implementazione.



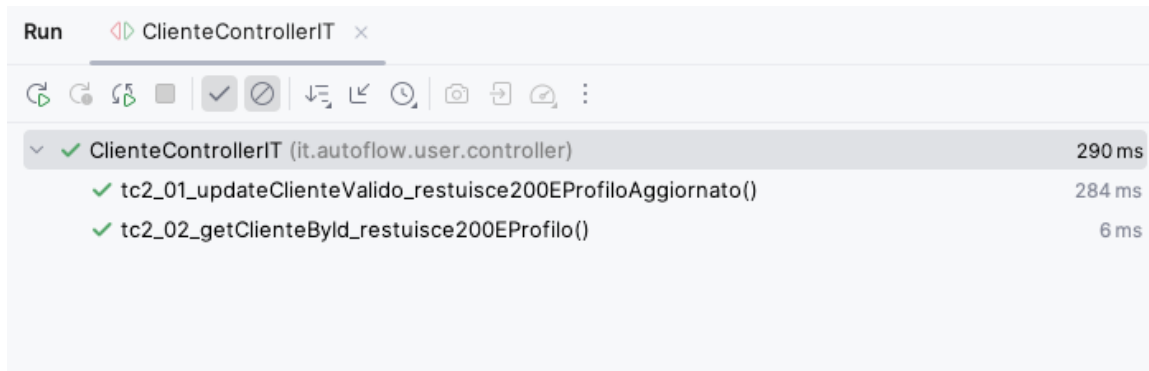
The screenshot shows a JUnit test runner window titled "Run" with a sub-header "ClienteServiceIT x". Below the header is a toolbar with various icons. The main area displays a list of test results for the package "it.autoflow.user.service". Each test is marked with a green checkmark and its execution time in milliseconds.

Test Name	Execution Time
✓ ClienteServiceIT (it.autoflow.user.service)	529 ms
✓ tc2_01_updateClienteValido_aggiornaProfiloInDb()	25 ms
✓ tc2_02_updateClientelnesistente_lanciaIllegalArgumentException()	41 ms
✓ tc2_03_deleteClienteEsistente_restituisceTrueERimuoveDaDb()	18 ms
✓ tc2_04_deleteClientelnesistente_restituisceFalse()	251 ms
✓ tc2_05_findAll_conClienti_restituisceListaNonVuota()	41 ms
✓ tc2_06_findAll_suDbVuoto_restituisceListaVuota()	33 ms
✓ tc2_07_getById_clienteEsistente_restituisceCliente()	43 ms
✓ tc2_08_getById_clientelnesistente_restituisceNull()	30 ms
✓ tc2_09_create_senzaUsernamePasswordRuolo_impostaValoriDiDefault()	23 ms
✓ tc2_10_update_modificaSoloCampiProfiloLasciandoAltriInvariati()	24 ms

Integration controller

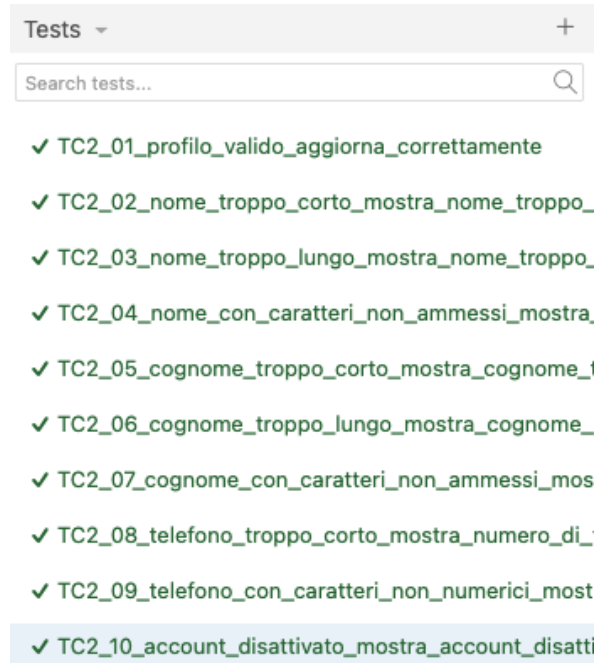
L'integrazione del controller è stata verificata tramite test con MockMvc sugli endpoint dedicati alla modifica del profilo e al cambio password, valutando la corretta gestione degli errori di validazione, la restituzione dei codici HTTP appropriati (200, 400 e 401) e la protezione garantita dai meccanismi di autenticazione. *(Si fa uso dello stesso discorso del TC1)*

Tutti i test sono **Passati**.



Test di sistema

Selenium IDE su Firefox ha permesso di verificare il comportamento dell'interfaccia nella sezione "Impostazioni account". Sono stati coperti scenari sia validi che invalidi per aggiornamento profilo e cambio password. L'intero set TC2-TC3 risulta **Passato**.



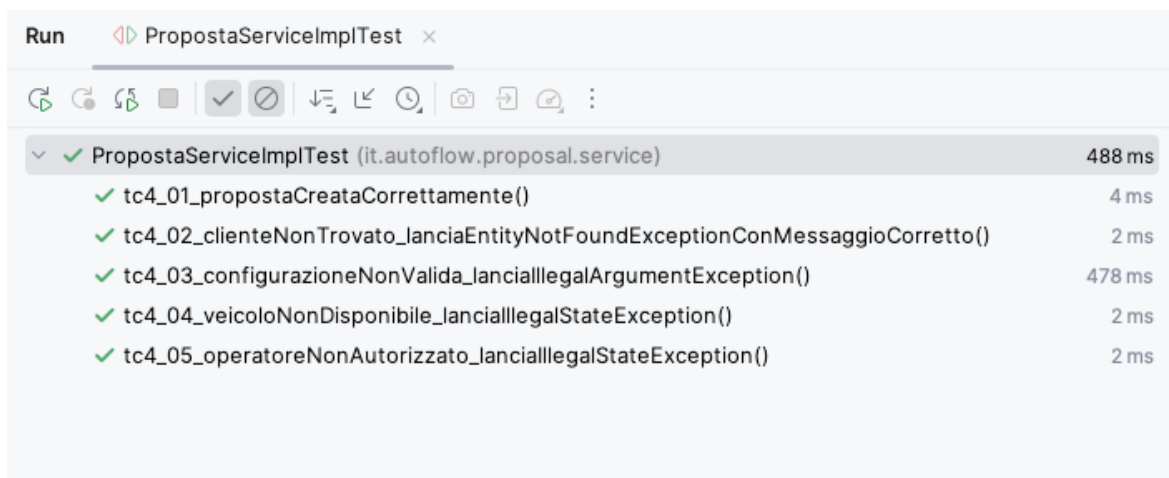
4.3 TC4 – Creazione proposta di vendita

Obiettivo: verificare la creazione di una proposta commerciale a partire da una configurazione, considerando cliente, configurazione e disponibilità veicolo.

Unit test

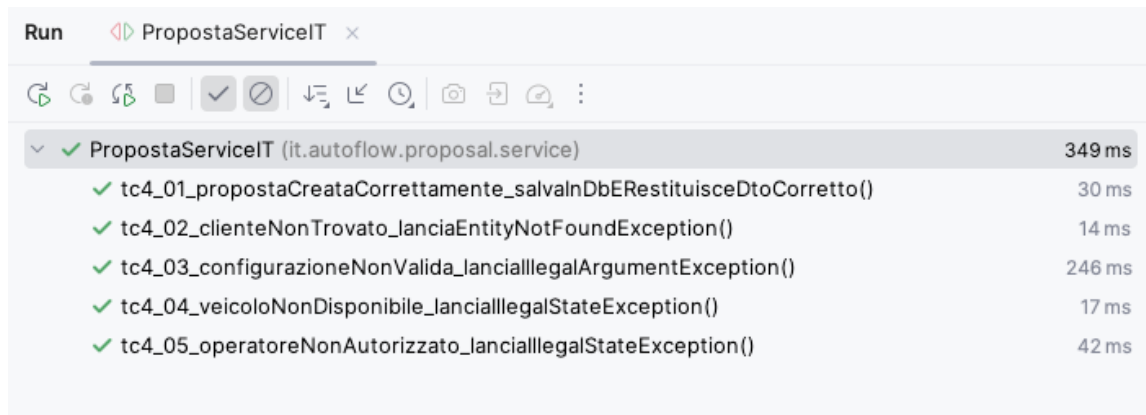
I test unitari del *PropostaService* hanno verificato il corretto rispetto della relazione tra cliente e configurazione, il controllo sullo stato del veicolo affinché risulti disponibile, la validazione dell'abilitazione dell'addetto vendite e, infine, la corretta creazione dell'entità Proposta.

Tutti i test TC4_01–TC4_05 risultano **Passati**.



Integration service con repository (H2)

In integrazione è stata verificata la persistenza dell'entità *Proposta* e la corretta gestione delle eccezioni per condizioni non valide. Tutti i test risultano **Passati**.



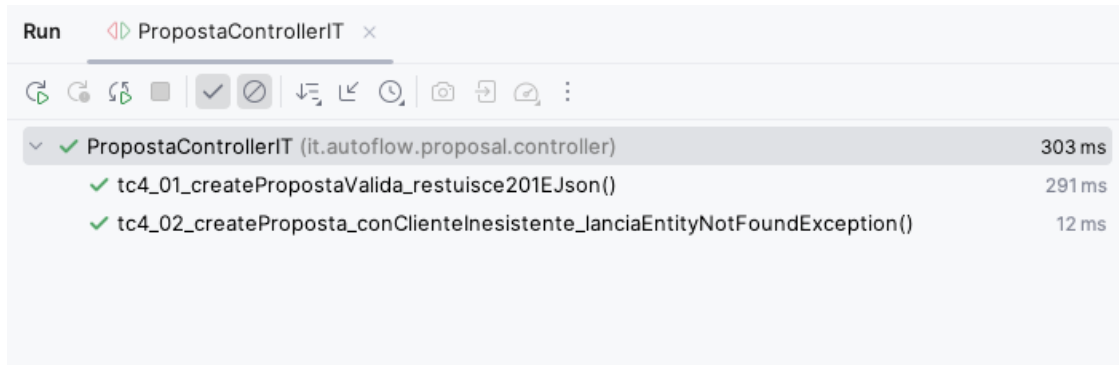
The screenshot shows the Run console of an IDE with the title bar 'Run' and 'PropostaServiceIT'. The console displays the execution of five tests, all of which passed successfully, indicated by green checkmarks. The tests are listed with their full names and the time taken for each to complete. The total time for the suite is 349 ms.

Test Name	Duration
✓ PropostaServiceIT (it.autoflow.proposal.service)	349 ms
✓ tc4_01_propostaCreataCorrettamente_salvaInDbERestituisceDtoCorretto()	30 ms
✓ tc4_02_clienteNonTrovato_lanciaEntityNotFoundException()	14 ms
✓ tc4_03_configurazioneNonValida_lanciaIllegalArgumentException()	246 ms
✓ tc4_04_veicoloNonDisponibile_lanciaIllegalStateException()	17 ms
✓ tc4_05_operatoreNonAutorizzato_lanciaIllegalStateException()	42 ms

Integration controller

MockMvc ha confermato la restituzione dello stato HTTP 201 in caso di creazione riuscita e la produzione di messaggi di errore coerenti con quanto definito nell'oracolo.

Tutti i test risultano **Passati**.



Test di sistema

Per i casi di test TC4 e TC5 sono stati eseguiti integralmente i test di unità e di integrazione; tuttavia, non è stato possibile automatizzare tramite Selenium alcuni scenari specifici, poiché l'interfaccia del sistema impedisce intenzionalmente la creazione di input o condizioni non previste. Tali limitazioni lato front-end impediscono la riproduzione artificiale di alcune situazioni di errore, già prevenute dal design applicativo.

4.4 TC5 - Generazione fattura PDF

Obiettivo: verificare la generazione di un documento PDF per proposte confermate, garantendo corretto calcolo importi, persistenza e salvataggio file.

Unit test

I test unitari hanno riguardato il *FatturaService* e hanno verificato che la fattura venga creata solo quando la proposta risulta confermata, che il totale calcolato sia corretto e che il PDF venga generato nel percorso previsto.

Tutti i test di unità sono **Passati** (*Tranne il tc5_04 che è stato risolto in seguito*).

The screenshot displays the test results in an IDE. The top panel shows a failed test: `tc5_04_spazioInsufficienteDuranteGenerazionePdf_J` with a duration of 459 ms. The error message is: `org.mockito.exceptions.misusing.UnnecessaryStubbingException: Unnecessary stubbings detected. Clean & maintainable test code requires zero unnecessary code. Following stubbings are unnecessary (click to navigate to relevant line of code): 1. -> at it.autoflow.invoice.service.FatturaServiceImplTest.tc5_04_spazioInsuffi Please remove unnecessary stubbings or use 'lenient' strictness. More info: javado at org.mockito.junit.jupiter.MockitoExtension.afterEach(MockitoExtension.java:`

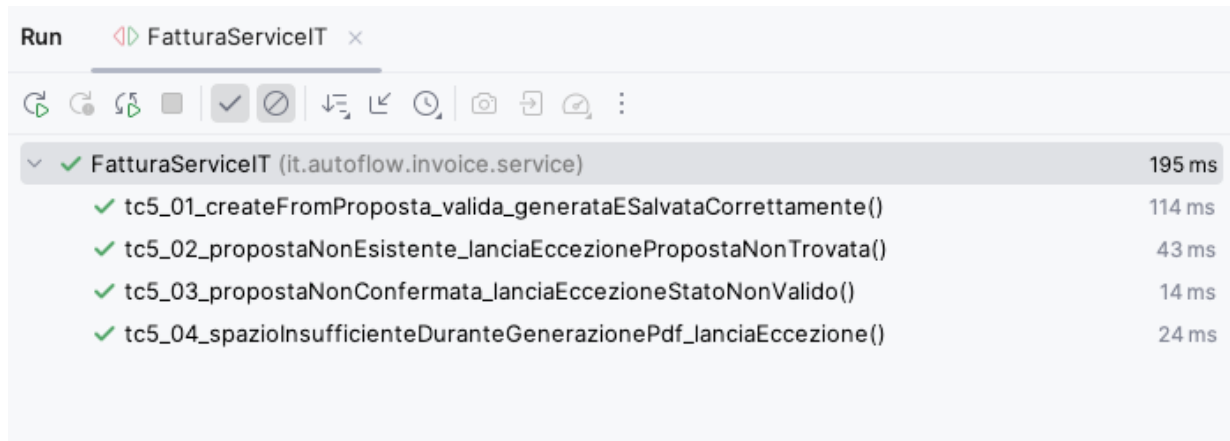
The bottom panel, titled "Run", shows a summary of all tests for `FatturaServiceImplTest` (it.autoflow.invoice.service) with a total duration of 502 ms. All tests passed:

Test Case	Duration
tc5_01_createFromProposta_valida_generataESalvataCorrettamente()	496 ms
tc5_02_propostaNonEsistente_lanciaEccezionePropostaNonTrovata()	2 ms
tc5_03_propostaNonConfermata_lanciaEccezioneStatoNonValido()	1 ms
tc5_04_spazioInsufficienteDuranteGenerazionePdf_lanciaEccezione()	3 ms

Integration service con repository (H2)

I test hanno confermato il corretto salvataggio della fattura con la relativa associazione alla proposta, il sollevamento dell'errore previsto nel caso in cui la proposta non risulti confermata e la corretta gestione delle eccezioni di I/O in presenza di problemi di scrittura.

Tutti i test risultano **Passati**.



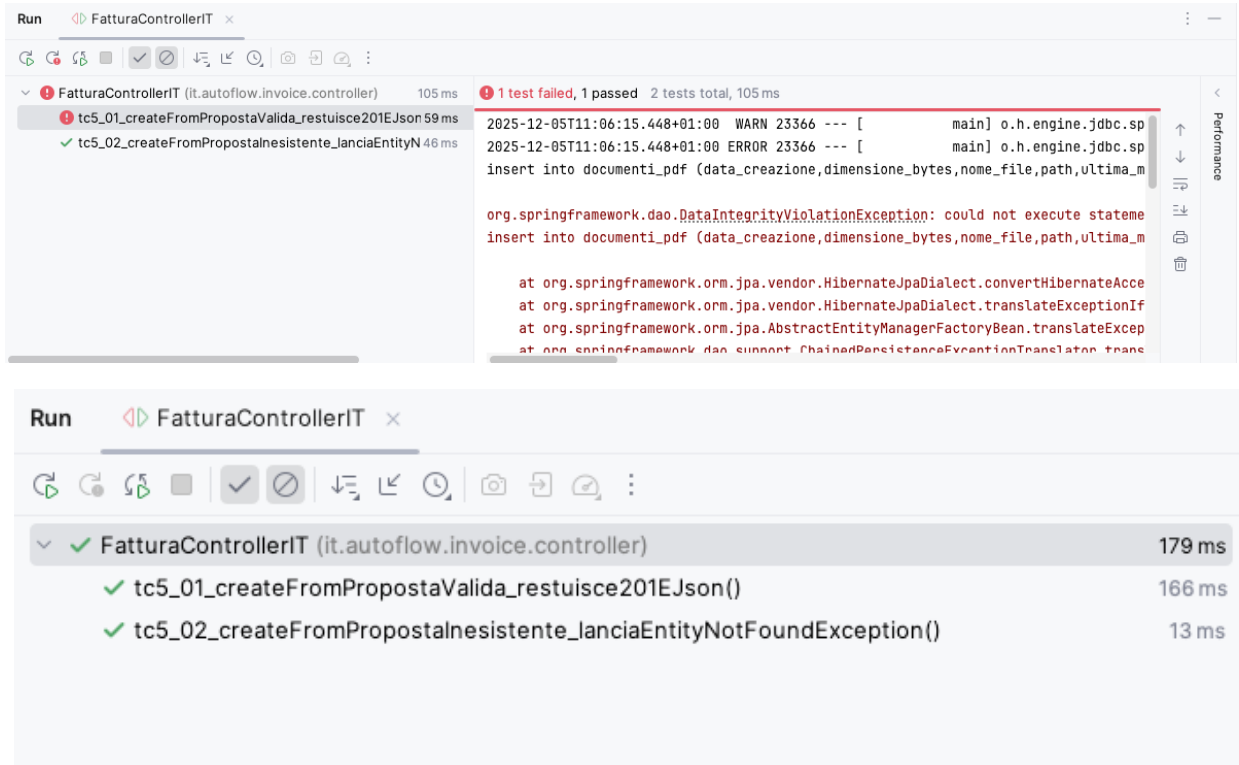
The screenshot shows the 'Run' console of an IDE. At the top, the title bar says 'Run' and 'FatturaServiceIT x'. Below the title bar is a toolbar with various icons for running and debugging. The main area of the console displays the test results for 'FatturaServiceIT (it.autoflow.invoice.service)'. The results are as follows:

Test Case	Duration
✓ FatturaServiceIT (it.autoflow.invoice.service)	195 ms
✓ tc5_01_createFromProposta_valida_generataESalvataCorrettamente()	114 ms
✓ tc5_02_propostaNonEsistente_lanciaEccezionePropostaNonTrovata()	43 ms
✓ tc5_03_propostaNonConfermata_lanciaEccezioneStatoNonValido()	14 ms
✓ tc5_04_spazioInsufficienteDuranteGenerazionePdf_lanciaEccezione()	24 ms

Integration controller

MockMvc ha verificato la restituzione dello stato HTTP 201 insieme al path logico del PDF prodotto e il comportamento corretto del sistema negli scenari di errore previsti.

Tutti i test sono **Passati** (*Tranne il tc5_01 che è stato risolto in seguito*).

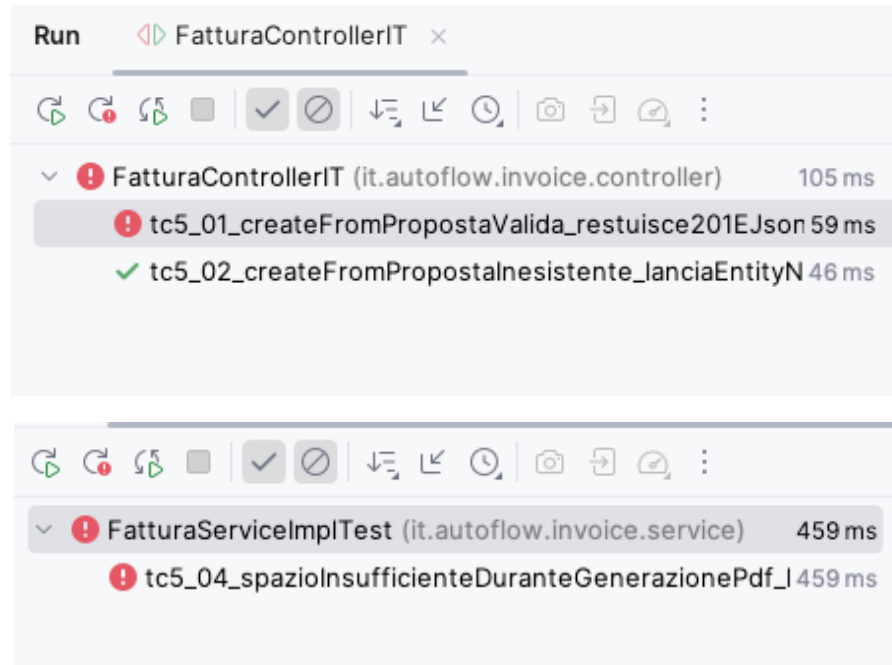


Test di sistema

Per i casi di test TC4 e TC5 sono stati eseguiti integralmente i test di unità e di integrazione; tuttavia, non è stato possibile automatizzare tramite Selenium alcuni scenari specifici, poiché l'interfaccia del sistema impedisce intenzionalmente la creazione di input o condizioni non previste. Tali limitazioni lato front-end impediscono la riproduzione artificiale di alcune situazioni di errore, già prevenute dal design applicativo.

5. Test Incident Report

Durante l'esecuzione del TC5 sono state rilevate dell'anomalie nei test tc5_01 (Controller), tc5_04 (ServiceImpl). Il log mostrava un'eccezione dovuta all'impossibilità di persistere l'entità. L'incidente provocato rende impossibile l'esecuzione dei due casi. Per correggere questa falla sarà effettuato l'opportuno debugging, seguito dall'esecuzione di tutta la suite di test.



6. Test Summary Report

Le attività di testing svolte sul progetto AutoFlow hanno permesso di validare in modo approfondito le principali funzionalità del sistema e di individuare tempestivamente eventuali anomalie derivate dallo sviluppo, evitando che tali errori si propagassero verso altri componenti del backend e, soprattutto, verso il frontend, che interagisce costantemente con il sistema tramite API REST. L'integrazione della suite di test nel processo di Continuous Integration ha garantito un controllo continuo sulle modifiche apportate al codice: ogni esecuzione fallita ha portato alla riesecuzione completa della suite, consentendo di verificare nuovamente la stabilità dell'applicazione.

La definizione dei casi di test si è basata sulla tecnica del Category Partition Testing, che ha permesso di individuare una serie completa di scenari significativi per le funzionalità considerate, assicurando una copertura adeguata dei possibili input e delle loro combinazioni. L'approccio adottato nell'esecuzione è stato quello "Sandwich", che ha previsto il collaudo parallelo sia dei servizi backend sia dei flussi utente delle interfacce. I test unitari hanno verificato l'esattezza della logica applicativa attraverso l'uso di Mockito, mentre i test di integrazione, eseguiti tramite il database H2 in-memory, hanno permesso di validare il comportamento del sistema in presenza di repository, transazioni e controller. I test di sistema, condotti tramite Selenium IDE su Firefox, hanno riprodotto i principali percorsi interattivi dell'utente finale, confermando la coerenza del comportamento dell'applicazione con quanto previsto nella TCS.

L'analisi della Code Coverage, effettuata attraverso l'IDE IntelliJ, ha fornito un quadro chiaro del livello di esercizio del codice da parte della suite di test. L'intero package `it.autoflow` ha raggiunto il 100% di copertura delle classi, accompagnato dal 46% sui metodi, il 63% sulle linee e il 49% sui branch. Il servizio dedicato alla gestione delle fatture presenta una copertura completa delle classi, con una copertura del 45% sui metodi, del 57% sulle linee e del 28% sui branch. Il servizio relativo alla creazione delle proposte presenta anch'esso una copertura totale delle classi, con una copertura del 33% sui metodi, del 52% sulle linee e del 40% sui branch. Per quanto riguarda la gestione degli utenti, il servizio dedicato ai clienti ha ottenuto un risultato particolarmente elevato, con una copertura del 62% sui metodi, del 94% sulle linee e del 79% sui branch.

Nel loro insieme, i risultati emersi confermano una buona affidabilità del sistema e costituiscono una base solida per futuri potenziamenti della suite di test.

Coverage autoflow in autoflow-backend				
Element ^				
	Class, %	Method, %	Line, %	Branch, %
✓ it.autoflow	100% (3/3)	46% (13/28)	63% (159/251)	49% (52/106)
invoice.service	100% (1/1)	45% (5/11)	57% (63/109)	28% (9/32)
© FatturaServiceImpl	100% (1/1)	45% (5/11)	57% (63/109)	28% (9/32)
proposal.service	100% (1/1)	33% (3/9)	52% (48/91)	40% (16/40)
© PropostaServiceImpl	100% (1/1)	33% (3/9)	52% (48/91)	40% (16/40)
user.service	100% (1/1)	62% (5/8)	94% (48/51)	79% (27/34)
© ClienteServiceImpl	100% (1/1)	62% (5/8)	94% (48/51)	79% (27/34)

7. Glossario

Termine	Definizione
Test unitario	Verifica del comportamento di singole unità di codice, generalmente metodi di servizio, mediante l'uso di mock per isolare le dipendenze.
Test di integrazione	Controllo del comportamento complessivo di più componenti del backend, comprendendo servizi, repository, transazioni e controller, spesso tramite database H2 in-memory.
Test di sistema	Validazione del comportamento percepito dall'utente finale attraverso l'interazione con il frontend, simulata grazie a Selenium IDE.
Oracolo di test	Risultato atteso definito nella specifica dei casi di test, utilizzato come riferimento per valutare l'esito della prova.
Continuous Integration	Processo automatizzato che esegue build, test e verifiche di qualità a ogni modifica del codice, garantendo stabilità e riduzione delle regressioni.
Code Coverage	Misura del grado di esercizio del codice sorgente da parte della suite di test, considerando classi, metodi, linee e branch.
Mock	Oggetto simulato utilizzato per isolare una dipendenza e verificare la logica di una singola unità in condizioni controllate.
H2 in-memory	Database leggero e transitorio utilizzato durante i test di integrazione, che consente esecuzioni rapide senza necessità di un'istanza esterna.
MockMvc	Strumento di Spring utilizzato per testare gli endpoint REST simulando richieste HTTP senza avviare un server reale.
Selenium IDE	Strumento usato per la creazione e l'esecuzione di script automatizzati che riproducono i comportamenti dell'utente nel browser.