

Università degli Studi di Salerno

Corso di Ingegneria del Software

AstroMark

Test Execution Report ed Incident Report

Versione 1.0



Data: 30/01/2025

Progetto: AstroMark	Versione: 1.0
Documento: Test Execution Report ed Incident Report	Data: 30/01/2025

Partecipanti:

Nome	Matricola
Giuseppe Cavallaro	0512116926
Mario Cosenza	0512116320
Mario Fasolino	0512116965
Giulio Sacrestano	0512116812

Scritto da:	Mario Cosenza

Revision History

Data	Versione	Descrizione	Autore
30/01/2025	1.0	Aggiunto report per i cinque test case definiti	Mario Cosenza

Indice

1.	Introduzione	4
2.	Relazione con altri documenti	4
3.	Test item transmittal report.....	4
4.	Test log.....	5
4.1	TC1	5
4.2	TC2-3	9
4.3	TC4	11
4.4	TC5.....	12
5.	Test Incident Report	15
6.	Test Summary Report.....	16
7.	Glossario.....	17

1. Introduzione

Il presente documento riporta l'esito di una delle esecuzioni della suite di test descritta nella specifica dei casi di test (Test Case Specification). Verranno evidenziati sia i comportamenti anomali, indicativi di potenziali difetti del sistema sotto test, sia i comportamenti conformi alle aspettative definite. Un test il cui risultato concorda con quanto previsto dall'oracolo di test viene classificato come "Passato". Al contrario, un test il cui risultato diverge da quanto previsto dall'oracolo viene classificato come "Fallito", segnalando la presenza di un'anomalia che necessita di ulteriori indagini e azioni correttive.

2. Relazione con altri documenti

Il presente progetto si basa sull'analisi e il confronto con piattaforme di gestione della didattica già consolidate e affermate nel settore, le quali hanno dimostrato notevole efficacia. 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 di unità e di integrazione, si utilizza un workflow definito tramite GitHub Actions. Tale workflow viene attivato automaticamente al rilevamento di una pull request indirizzata al branch main. Qualora si desideri eseguire i test in locale, seguendo le indicazioni contenute nella presente documentazione, è necessario avviare l'immagine Docker di PostgreSQL 17.2. Successivamente, accedendo al package `src/test/java`, è possibile eseguire la suite di test tramite un IDE quale IntelliJ, oppure mediante il comando `mvn test`.

Per l'esecuzione dei test di sistema con Selenium IDE, è richiesta l'importazione del file `AstroMark.side` nella directory `src/script`. Si precisa che non tutti i test di sistema possono essere eseguiti in modalità completamente automatica, in quanto alcuni di essi richiedono l'interazione manuale dell'utente, ad esempio per effettuare il login a seguito della modifica delle credenziali di accesso prevista in TC1.

4. Test log

Di seguito sono presentati i risultati delle attività di testing, relativi ai casi di test definiti nella specifica dei casi di test (Test Case Specification). Per ciascun test eseguito, viene fornito il relativo esito, accompagnato da una breve descrizione della tipologia di test implementata. Nell'esecuzione dei test, ci si è avvalsi di diverse librerie e framework, tra cui JUnit 5 per la gestione dei test unitari, Mockito per la creazione di mock di dipendenze esterne, Testcontainers per l'esecuzione di test di integrazione con dipendenze containerizzate e Faker per la generazione di dati di test realistici ma casuali.

4.1 TC1

Il Test Case 1 è relativo al testing della funzionalità del login e della creazione del token JWT. Di seguito sono mostrati i risultati di test di unità, integrazione e di sistema.

Unit test

I test unitari condotti sul servizio di autenticazione, in particolare sul metodo 'login' della classe AuthenticationServiceImpl, hanno prodotto esiti positivi essendo il risultato conforme con quello atteso. L'utilizzo di oggetti mock per i repository ha permesso di isolare la logica del metodo e verificarne il corretto funzionamento. I test sono dunque "Passati".

✓ AuthenticationServiceTest

- ✓ tc1_01()
- ✓ tc1_02()
- ✓ tc1_03()
- ✓ tc1_04()
- ✓ tc1_05()
- ✓ tc1_06()
- ✓ tc1_07()
- ✓ tc1_08()
- ✓ tc1_09()
- ✓ tc1_10()
- ✓ tc1_11()
- ✓ tc1_12()
- ✓ tc1_13()
- ✓ tc1_14()
- ✓ tc1_15()

Integration service con repository

In seguito al test di unità di AuthenticationService è stato effettuato il test di integrazione, in questo caso il contesto utilizzato è quello dell'intera applicazione, definito tramite l'annotazione @SpringBootTest.

Per garantire il massimo isolamento, tra l'esecuzione dei diversi test, sono utilizzate entità differenti per evitare che la modifica di una di queste possa avere effetti collaterali sugli altri test. Il database utilizzato è lo stesso di quello di produzione, eseguito tramite Docker. Per questi test è stato utilizzato JUnit 5 e l'utente testato è "Student". Anche in questo caso i test sono "Passati".

```
✓ AuthenticationServiceIntegrationTest
  ✓ tc1_01()
  ✓ tc1_02()
  ✓ tc1_03()
  ✓ tc1_04()
  ✓ tc1_05()
  ✓ tc1_06()
  ✓ tc1_07()
  ✓ tc1_08()
  ✓ tc1_09()
  ✓ tc1_10()
  ✓ tc1_11()
  ✓ tc1_12()
  ✓ tc1_13()
  ✓ tc1_14()
  ✓ tc1_15()
```

Integration controller

Per la validazione dell'endpoint `/api/auth/login` di `AuthController`, sono stati impiegati gli strumenti di testing `MockMvc`, `JUnit` e `Testcontainers`. Utilizzando il metodo `HTTP POST`, è stata simulata una richiesta di autenticazione.

L'output atteso, in formato `JSON`, è stato confrontato con quello effettivo per verificare la corretta funzionalità del controller. Tali test hanno consentito di convalidare l'integrazione tra i vari layer dell'applicazione `Spring Boot`. Tutti i test sono stati superati con successo.

✓ `AuthControllerTest`

✓ `tc1_01()`

✓ `tc1_02()`

✓ `tc1_03()`

✓ `tc1_04()`

✓ `tc1_05()`

✓ `tc1_06()`

✓ `tc1_07()`

✓ `tc1_08()`

✓ `tc1_09()`

✓ `tc1_10()`

✓ `tc1_11()`

✓ `tc1_12()`

✓ `tc1_13()`

✓ `tc1_14()`

✓ `tc1_15()`

Test di Sistema

I test di sistema della funzionalità login sono stati realizzati tramite Selenium IDE. Alcune funzionalità per specifici controlli introdotti in TypeScript non sono testabili seguendo i test frame precedentemente definiti. Il test di questi aspetti è, comunque, garantito dal testing del controller essendo il frontend indipendente dall'implementazione del backend. Di seguito sono riportati gli screenshot dell'esecuzione della suite denominata TC1.

▼ TC1*

✓ TC1_03*

✓ TC1_04*

✓ TC1_06*

✓ TC1_07*

✓ TC1_08*

✓ TC1_09*

✓ TC1_10*

✓ TC1_11*

✓ TC1_12*

✓ TC1_13*

✓ TC1_14*

✓ TC1_15*

Risultato test TC1

Esempio di test registrato tramite Selenium

	Command	Target	Value
1.	✓ open	http://localhost:5173/	
2.	✓ set window size	1175x1258	
3.	✓ click	css= MuiButton-root	
4.	✓ mouse over	css= MuiButton-text	
5.	✓ click	id= r0.	
6.	✓ type	id= r0.	SS12345
7.	✓ click	id= r1.	
8.	✓ type	id= r1.	pluto pippo
9.	✓ click	id= r2.	
10.	✓ mouse over	css= MuiButtonBase-root:nth-child(2)	
11.	✓ mouse out	css= MuiButtonBase-root:nth-child(2)	
12.	✓ type	id= r2.	Pluto123!
13.	✓ click	css= MuiButton-contained	
14.	✓ assert element present	id=errorLogin	

4.2 TC2-3

I test case 2 e 3 sono relativi al testing delle funzionalità di cambio password e aggiornamento indirizzo di residenza. I log dei test sono riportati insieme ad una breve descrizione di come i test sono stati condotti.

Unit test

I test di unità, condotti sul servizio di cambio password e aggiornamento indirizzo di residenza, hanno prodotto esiti positivi essendo il risultato conforme con quello atteso. Entrambi i metodi sono implementati in `SchoolUserServiceImpl` e come per TC1 è stato utilizzato Mockito, Faker e JUnit. Come utente scelto per il test è stato usato un mock di `Student`.

Anche i test di unità di TC2 e TC3 risultano "Passati".

```
✓ SchoolUserServiceTest
  ✓ tc2_01()
  ✓ tc2_02()
  ✓ tc2_03()
  ✓ tc2_04()
  ✓ tc2_05()
  ✓ tc3_01()
  ✓ tc3_02()
  ✓ tc3_03()
  ✓ tc3_04()
  ✓ tc3_05()
```

Integration service con repository

Il test di integrazione di `SchoolUserServiceImpl` è stato effettuato utilizzando come contesto quella dell'intera applicazione ed è stata assegnata un'authority con ruolo `Student` all'entità oggetto del cambio password e aggiornamento indirizzo, essendo presente un controllo del ruolo al livello del Service. I test sono stati effettuati con l'ausilio degli stessi framework e librerie citate per il TC1. I test di integrazione di TC2 e TC3 sono "Passati".

```
✓ SchoolUserServiceIntegrationTest
  ✓ tc2_01()
  ✓ tc2_02()
  ✓ tc2_03()
  ✓ tc2_04()
  ✓ tc2_05()
  ✓ tc3_01()
  ✓ tc3_02()
  ✓ tc3_03()
  ✓ tc3_04()
  ✓ tc3_05()
```

Integration controller

Per il test degli endpoint `/api/school-users/address` e `/api/school-users/preferences` di `SchoolUserController`, sono stati impiegati gli strumenti di testing `MockMvc`, `JUnit` e `Testcontainers`. Utilizzando il metodo `HTTP POST`, è stata effettuata una richiesta di autenticazione, il token ottenuto è stato utilizzato come valore da inserire nel header. `Authentication` per le successive richieste `HTTP PATCH`.
Tutti i test sono stati superati con successo.



Test di Sistema

I test di sistema della funzionalità di aggiornamento password e cambio indirizzo di residenza sono stati realizzati tramite `Selenium IDE`. La pagina di riferimento è quella delle impostazioni utente. Per l'esecuzione della test suite, relativa all'aggiornamento password, è necessario un intervento manuale per autenticare nuovamente l'utente poiché è effettuato, in seguito, automaticamente il logout dell'utente. I test risultano tutti "Passati".
Di seguito sono riportati gli screenshot dell'esecuzione delle suite denominate TC2 e TC3.

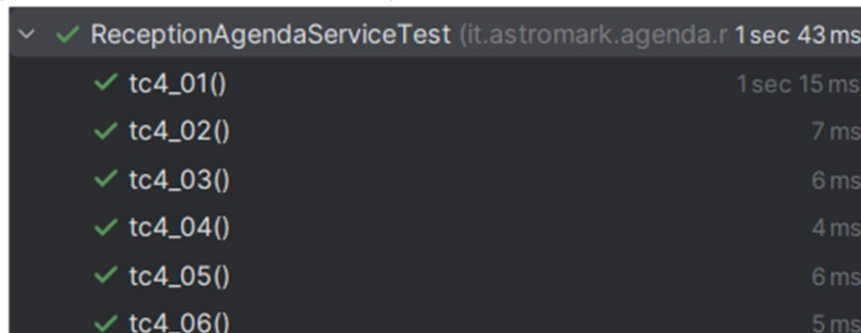


4.3 TC4

Il Test Case 4 è stato definito per testare la funzionalità di prenotazione del ricevimento da parte di un genitore. Questi test mirano a evidenziare anomalie nei controlli, sia a livello di autenticazione che di situazioni limite, come il tentativo di prenotare slot pieni.

Unit test

I test unitari condotti sul servizio di prenotazione del ricevimento, in particolare sul metodo "book" della classe ReceptionAgendaServiceImpl, hanno prodotto esiti positivi essendo il risultato conforme con quello atteso. I test sono dunque "Passati".

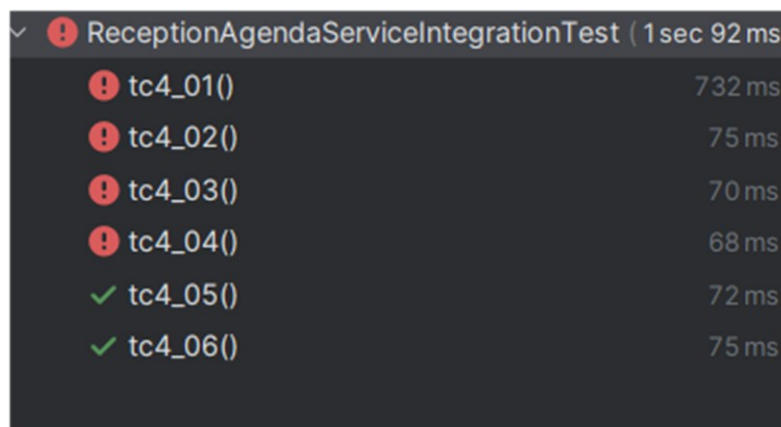


A screenshot of a JUnit test runner showing the results for the 'ReceptionAgendaServiceTest' class. The header indicates the class is from 'it.astromark.agenda.r' and the total execution time is 1 sec 43 ms. Below the header, six individual test methods are listed, each preceded by a green checkmark icon, indicating they all passed. The methods are tc4_01(), tc4_02(), tc4_03(), tc4_04(), tc4_05(), and tc4_06(). To the right of each method name, the execution time for that specific test is displayed.

✓	ReceptionAgendaServiceTest (it.astromark.agenda.r	1 sec 43 ms
✓	tc4_01()	1 sec 15 ms
✓	tc4_02()	7 ms
✓	tc4_03()	6 ms
✓	tc4_04()	4 ms
✓	tc4_05()	6 ms
✓	tc4_06()	5 ms

Integration service con repository

Durante il testing di integrazione è stato riscontrato il fallimento di tc4_01, tc4_02, tc4_03, tc4_4. Dall'analisi dei log l'eccezione prodotta è "fail to persist entity", eccezione legata a SpringData Jpa e Hibernate. Il testing degli altri layer è stato sospeso. Le azioni intraprese per la correzione del bug sono descritte nel test incident report.



A screenshot of a JUnit test runner showing the results for the 'ReceptionAgendaServiceIntegrationTest' class. The header indicates the class is from 'it.astromark.agenda.r' and the total execution time is 1 sec 92 ms. Below the header, six individual test methods are listed. The first four methods (tc4_01(), tc4_02(), tc4_03(), and tc4_04()) are preceded by a red exclamation mark icon, indicating they failed. The last two methods (tc4_05() and tc4_06()) are preceded by a green checkmark icon, indicating they passed. To the right of each method name, the execution time for that specific test is displayed.

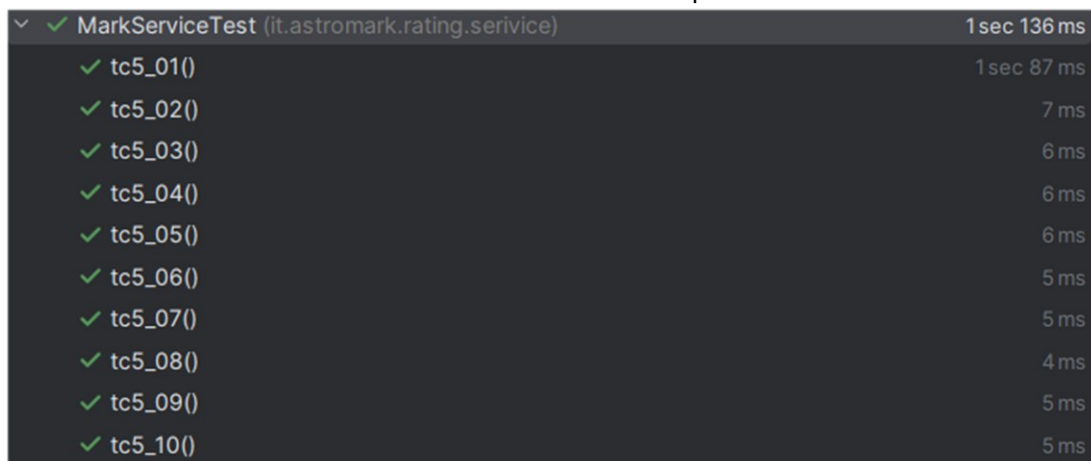
✓	ReceptionAgendaServiceIntegrationTest (1 sec 92 ms
!	tc4_01()	732 ms
!	tc4_02()	75 ms
!	tc4_03()	70 ms
!	tc4_04()	68 ms
✓	tc4_05()	72 ms
✓	tc4_06()	75 ms

4.4 TC5

Il Test Case 5 è stato definito per consentire il testing della funzionalità di inserimento valutazioni del docente. Il testing è stato condotto per valutare la presenza di anomalie nei controlli, partendo dai valori passati dal frontend e convertiti da JSON a un corrispondente DTO Java.

Unit test

I test unitari condotti sul servizio di inserimento mark, della classe MarkServiceImpl, hanno prodotto esiti positivi e conformi all'oracolo. L'utente utilizzato per il test è "Teacher", anche questo definito tramite MockUser. I test di unità sono dunque "Passati".

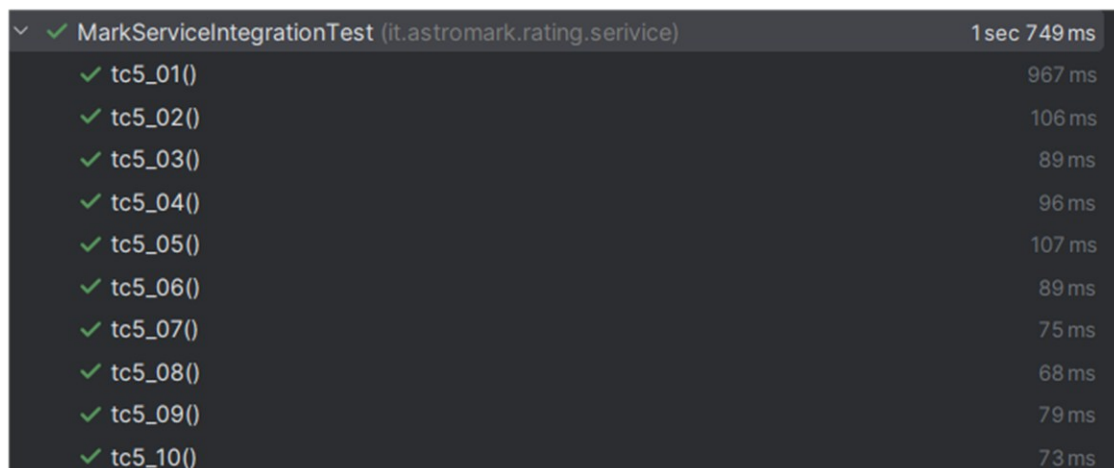


A screenshot of a JUnit test runner interface showing the results for the `MarkServiceTest` class. The interface has a dark background. At the top, it shows the class name `MarkServiceTest (it.astromark.rating.service)` with a green checkmark and a total execution time of `1 sec 136 ms`. Below this, a list of ten test methods is shown, each with a green checkmark and its execution time. The methods are `tc5_01()` through `tc5_10()`. The times range from 4 ms to 87 ms.

Test Method	Execution Time
✓ tc5_01()	1 sec 87 ms
✓ tc5_02()	7 ms
✓ tc5_03()	6 ms
✓ tc5_04()	6 ms
✓ tc5_05()	6 ms
✓ tc5_06()	5 ms
✓ tc5_07()	5 ms
✓ tc5_08()	4 ms
✓ tc5_09()	5 ms
✓ tc5_10()	5 ms

Integration service con repository

Il test di integrazione di MarkServiceImpl è stato effettuato utilizzando come contesto quella dell'intera applicazione ed è stata assegnata un'autorità con ruolo "Teacher", per il docente che intende effettuare l'inserimento delle valutazioni. I test sono stati effettuati con l'ausilio degli stessi framework e librerie citate per il TC1. I test di integrazione di TC5 sono "Passati".



A screenshot of a JUnit test runner interface showing the results for the `MarkServiceIntegrationTest` class. The interface has a dark background. At the top, it shows the class name `MarkServiceIntegrationTest (it.astromark.rating.service)` with a green checkmark and a total execution time of `1 sec 749 ms`. Below this, a list of ten test methods is shown, each with a green checkmark and its execution time. The methods are `tc5_01()` through `tc5_10()`. The times range from 68 ms to 106 ms.

Test Method	Execution Time
✓ tc5_01()	967 ms
✓ tc5_02()	106 ms
✓ tc5_03()	89 ms
✓ tc5_04()	96 ms
✓ tc5_05()	107 ms
✓ tc5_06()	89 ms
✓ tc5_07()	75 ms
✓ tc5_08()	68 ms
✓ tc5_09()	79 ms
✓ tc5_10()	73 ms

Integration controller

Per il test dell'endpoint /api/students/marks di MarkController, utilizzando il metodo HTTP POST, è stata effettuata una richiesta di autenticazione, il token ottenuto è stato utilizzato come valore da inserire nel header Authentication per le successive richieste HTTP POST all'endpoint per l'inserimento delle valutazioni. Tutti i test sono stati superati con successo.

✓ MarkControllerTest (it.astromark.rating.controller)	1 sec 748 ms
✓ tc5_01()	967 ms
✓ tc5_02()	115 ms
✓ tc5_03()	100 ms
✓ tc5_04()	93 ms
✓ tc5_05()	88 ms
✓ tc5_06()	85 ms
✓ tc5_07()	74 ms
✓ tc5_08()	68 ms
✓ tc5_09()	83 ms
✓ tc5_10()	75 ms

Test di Sistema

I test di sistema della funzionalità di inserimento voti sono stati realizzati tramite Selenium IDE. Alcune funzionalità per specifici controlli introdotti in TypeScript non sono testabili seguendo i test frame precedente definiti. Il test di questi aspetti è comunque garantito dal testing del controller, essendo il frontend indipendente dall'implementazione del backend. Di seguito sono riportati gli screenshot dell'esecuzione della suite TC5.

▼ TC5*

✓ TC5_01*

✓ TC5_02*

✓ TC5_04*

✓ TC5_03*

✓ TC5_05*

✓ TC5_06*

✓ TC5_07*

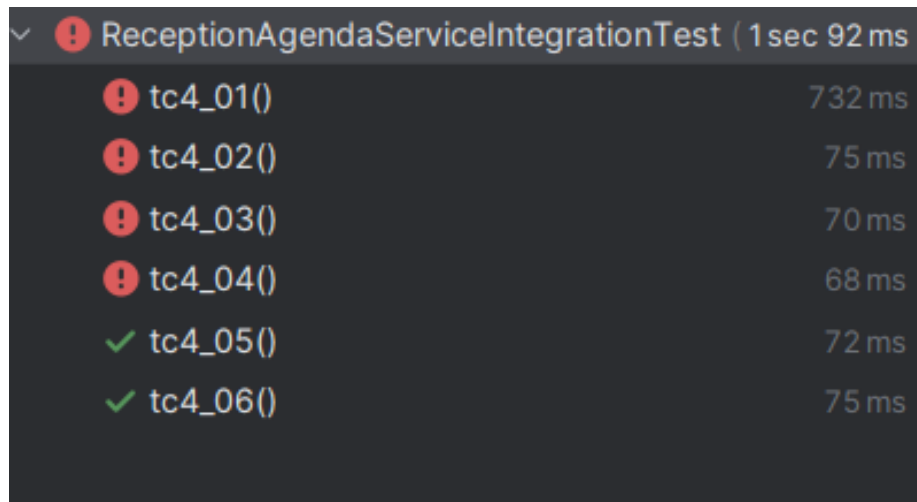
Risultato test TC5

Esempio di test registrato tramite Selenium

	Command	Target
▶ TC1*	1 ✓ open	http://localhost:5173/parenticevimento
▶ TC2	2 ✓ set window size	1175x1250
▶ TC3	3 ✓ mouse over	name=select_row
▼ TC4*	4 ✓ click	name=select_row
✓ TC4_01*	5 ✓ mouse out	name=select_row
✓ TC4_02*	6 ✓ mouse over	css= MuButton-root
✓ TC4_03*	7 ✓ mouse over	css= MuButton-root
✓ TC4_04*	8 ✓ click	css= MuButton-root
✓ TC4_05*	9 ✓ mouse out	css= MuButton-root
✓ TC4_06*	10 ✓ assert element present	css= MuDataGrid-columnHeader-alignCenter .MuDataGrid-columnHeaderTitleContainerContent

5. Test Incident Report

Durante l'esecuzione del TC4 sono state rilevate dell'anomalie nei test tc4_01, tc4_02, tc4_04, tc4_04. Il log mostrava un'eccezione dovuta all'impossibilità di persistere l'entità ReceptionBooking. L'incidente provocato rende impossibile prenotare il ricevimento docenti ai genitori. Per correggere questa falla sarà effettuato l'opportuno debugging, seguito dall'esecuzione di tutta la suite di test.



The screenshot shows a test execution log for 'ReceptionAgendaServiceIntegrationTest' which took 1 second and 92 milliseconds. The log lists six test cases: tc4_01(), tc4_02(), tc4_03(), tc4_04(), tc4_05(), and tc4_06(). The first four tests are marked with a red exclamation mark icon, indicating they failed. The last two tests, tc4_05() and tc4_06(), are marked with a green checkmark icon, indicating they passed. The execution time for each test is listed in milliseconds.

✓ ! ReceptionAgendaServiceIntegrationTest (1sec 92 ms	
! tc4_01()	732 ms
! tc4_02()	75 ms
! tc4_03()	70 ms
! tc4_04()	68 ms
✓ tc4_05()	72 ms
✓ tc4_06()	75 ms

6. Test Summary Report

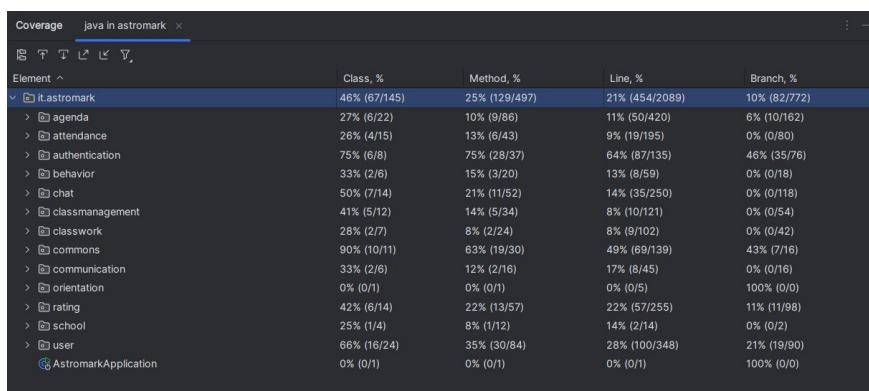
Le attività di testing hanno consentito l'individuazione di anomalie introdotte durante la fase di sviluppo, prevenendone la propagazione in altre componenti del codice e, in particolare, al frontend, dove i servizi sono esposti e utilizzati tramite API REST. L'adozione di strumenti quali SonarQube ha contribuito alla riduzione di problematiche di manutenibilità e alla mitigazione di potenziali vulnerabilità di sicurezza, queste ultime individuate anche grazie al code scanning integrato in GitHub. In seguito al rilevamento di un fallimento in uno dei test, è stata rieseguita l'intera suite sfruttando il workflow di Continuous Integration (CI) appositamente configurato.

Per garantire un'adeguata copertura e completezza dei test, è stata impiegata la tecnica del Category Partition Testing, che ha permesso di definire un insieme esaustivo di casi di test per le funzionalità coperte.

I test sono stati eseguiti seguendo un approccio "Sandwich", con i test delle pagine React condotti in parallelo allo sviluppo del backend Spring Boot. Per i test di integrazione, in particolare quelli che interagiscono con database o altri servizi esterni, è stata utilizzata la libreria Testcontainers con i seguenti vantaggi:

- **Isolamento degli ambienti di test:** ogni test viene eseguito in un ambiente pulito e isolato, prevenendo interferenze tra test diversi e garantendo risultati consistenti.
- **Test di integrazione più affidabili:** Testcontainers permette di testare l'integrazione tra l'applicazione Spring Boot e le sue dipendenze reali, aumentando l'affidabilità dei test di integrazione.

L'analisi della Code Coverage, effettuata direttamente all'interno dell'IDE IntelliJ, ha fornito preziose informazioni sullo stato di avanzamento del testing. In particolare, è stato possibile osservare una buona copertura in alcune aree del progetto, come ad esempio nel package commons, che raggiunge il 90% di coverage per le classi e il 63% per i metodi, e nel package authentication, con il 75% di coverage sia per le classi che per i metodi. Questo indica che le funzionalità centrali e di autenticazione sono state testate in modo approfondito.



Element	Class, %	Method, %	Line, %	Branch, %
it.astromark	46% (67/145)	25% (129/497)	21% (454/2089)	10% (82/772)
agenda	27% (6/22)	10% (9/86)	11% (50/420)	6% (10/162)
attendance	26% (4/15)	13% (6/43)	9% (19/195)	0% (0/80)
authentication	75% (6/8)	75% (28/37)	64% (87/135)	46% (35/76)
behavior	33% (2/6)	15% (3/20)	13% (8/59)	0% (0/18)
chat	50% (7/14)	21% (11/52)	14% (35/250)	0% (0/118)
classmanagement	41% (5/12)	14% (5/34)	8% (10/121)	0% (0/54)
classwork	28% (2/7)	8% (2/24)	8% (9/102)	0% (0/42)
commons	90% (10/11)	63% (19/30)	49% (69/139)	43% (7/16)
communication	33% (2/6)	12% (2/16)	17% (8/45)	0% (0/16)
orientation	0% (0/1)	0% (0/1)	0% (0/5)	100% (0/0)
rating	42% (6/14)	22% (13/57)	22% (57/255)	11% (11/98)
school	25% (1/4)	8% (1/12)	14% (2/14)	0% (0/2)
user	66% (16/24)	35% (30/84)	28% (100/348)	21% (19/90)
AstromarkApplication	0% (0/1)	0% (0/1)	0% (0/1)	100% (0/0)

7. Glossario

Termine	Descrizione
Test unitario	Verifica la correttezza di una singola unità di codice (es. una funzione o un metodo).
Test di integrazione	Verifica l'interazione tra diversi componenti di un sistema.
Test di sistema	Verifica il funzionamento completo del sistema.
Caso di test	Un insieme di condizioni o variabili usate per valutare un sistema.
Suite di test	Una raccolta di casi di test.
Scenario di test	Una sequenza di azioni eseguite per verificare una specifica funzionalità.
JUnit 5	Framework Java per la creazione e l'esecuzione di test unitari.
Mockito	Framework Java per la creazione di oggetti fittizi (mock) durante i test.
Testcontainers	Libreria Java che fornisce contenitori leggeri e usa e getta di database, browser e altre dipendenze per i test.
Faker	Libreria per generare dati fittizi.
Selenium IDE	Strumento per registrare e riprodurre test di applicazioni web.
MockMvc	Framework di testing per Spring MVC.
GitHub Actions	Piattaforma per l'integrazione continua e la consegna continua.
Pull request	Richiesta di unire delle modifiche al codice sorgente principale.
Docker	Piattaforma per la creazione e l'esecuzione di contenitori.
PostgreSQL	Sistema di gestione di database relazionale open-source.
JWT (JSON Web Token)	Standard per la trasmissione sicura di informazioni tra parti sotto forma di oggetto JSON.
Copertura del codice	Misura che indica la porzione di codice effettivamente testata.