UNIVERSITÀ DEGLI STUDI DI SALERNO
DIPARTIMENTO DI INFORMATICA

Ingegneria del Software

UniClass - Test Execution Report (TER)

Baseline Assessment & Post-Refactoring Validation

Versione 1.1

DATA: 15/02/2026

Partecipanti:

Nome	Matricola
Cammarota Lucageneroso	NF22500053
Sabetta Giuseppe	NF22500155

Scritto da:	Cammarota Lucageneroso, Sabetta Giuseppe

Revision History

Data	Versione	Descrizione	Autore
15/02/2026	1.1	Aggiunta dell'analisi del Testing non Funzionale	GS
13/02/2026	1.0	Baseline Analysis e Validazione Nuova Suite	LC

Indice

1 Introduzione

Il presente documento costituisce il *Test Execution Report* (TER) del sistema *UniClass*. In accordo con lo standard **ISO/IEC/IEEE 29119-3**, esso documenta i risultati dell'esecuzione dei test in due fasi distinte: la validazione della Baseline (stato "As-Is") e la verifica della nuova suite post-manutenzione (stato "To-Be"), a seguito del refactoring architetturale del modulo Utenti.

1.1 Scopo del Documento

L'obiettivo è certificare che l'intervento di manutenzione evolutiva (passaggio da ereditarietà a composizione/ruoli) e correttiva (miglioramento testabilità e isolamento) sia avvenuto senza introdurre regressioni funzionali, garantendo la correttezza delle nuove implementazioni.
Si potranno analizzare le differenze dal punto di vista funzionale e non funzionale tra la baseline e il medesimo progetto post-refactoring, comprendendone le motivazioni, indipendentemente dalla qualità della nuova piattaforma.

1.2 Riferimenti

- **ISO/IEC/IEEE 29119-3**: Standard internazionale per la documentazione del software testing.
- **Change Request (CR) v1.0**: Definizione del refactoring del modello dati Utenti.
- **Impact Analysis Report v1.0**: Identificazione del *Candidate Impact Set* (CIS).
- **Maven Surefire Reports**: Log tecnici delle esecuzioni (Baseline e NewTestSuite).

2 Test Execution Summary

L'esecuzione dei test è stata condotta in ambiente controllato al fine di valutare la conformità funzionale del sistema UniClass dopo l'intervento di manutenzione sul modulo Utenti.

2.1 Test Environment - Funzionale

Il seguente ambiente di esecuzione è stato utilizzato e misurato adeguatamente per determinare il tempo di esecuzione dei test funzionali. Nonostante la mancata necessità della misurazione di metriche durante il Testing Funzionale, si è deciso comunque di riportare l'ambiente per aumentare la riproducibilità e il dettaglio del lato tecnico della fase specificata.

- **Build testata**: UniClass v1.0 (baseline e post-refactoring)
- **Framework di test**: JUnit 5
- **Tool di build**: Maven

- **Plugin di reporting**: Maven Surefire
- **Sistema operativo**: Windows
- **JVM**: 21

2.2 Test Environment - Non Funzionale

L'ambiente di esecuzione per il Testing non funzionale, presente anche nella baseline del Test Plan, privo di servizi esterni o deleteri per un'accurata e ottimale misurazione, è il seguente:

- **OS**: Nobara Linux 43 (KDE Plasma Desktop Edition) x86_64
- **Kernel**: Linux 6.18.7-200.nobara.fc43.x86_64
- **CPU**: AMD Ryzen 5 3500 - 6 core @4.12 GHz
- **GPU**: AMD Radeon RX 5500XT
- **Memory (Physical)**: 16 GB DDR4
- **Memory (Swap-Digital)**: 8 GB

2.3 Tested Configuration

Sistema	UniClass
Versione Baseline	v1.0 As-Is
Versione Post-Manutenzione	v1.0 To-Be
Modulo oggetto di intervento	Utenti

2.4 Execution Outcome

L'esecuzione complessiva della test suite ha soddisfatto i criteri di completamento definiti nel Test Plan. Tuttavia ha evidenziato failure diffuse nella test suite legacy, successivamente analizzate nella Sezione Incident Summary.

3 Risultati della Baseline (Pre-Refactoring) - Funzionale

In questa sezione vengono presentati i risultati dell'esecuzione della test suite sulla versione legacy del sistema, prima dell'applicazione delle modifiche.

3.1 Sintesi Quantitativa

L'esecuzione ha coinvolto l'intera suite di regressione originale.

Metrica	Valore	Note
Totale Test Eseguiti	969	Copertura regressione completa
Test Superati (Pass)	969	100% Success Rate
Tempo Totale Esecuzione	7.199 s	Media ∼7.4ms/test

Tabella 2: Riepilogo Esecuzione Baseline (Fonte: Maven Surefire Report)

3.2 Analisi della Copertura per Modulo

La baseline mostrava una forte concentrazione di test nel modulo *Orari*, con una strategia basata prevalentemente su test di unità "leggeri", spesso dipendenti da stati condivisi o da implementazioni mock parziali.

- **Modulo Orari:** 277 test nel Model, 189 nei Service.
- **Modulo Utenti:** Copertura frammentata attraverso ereditarietà (test su `Studente`, `Docente` separati).

4 Risultati della Baseline (Pre-Refactoring) Non Funzionale

Prima di determinare la strategia di adeguamento della Test Suite, funzionale e non, è importante specificare nuovamente, poichè già presente nel Test Plan, la baseline non funzionale a livello unitario e sistemico. Si identificherà la baseline riguardante le metriche d'uso di JMH (Java Microbenchmark Harness) e di Apache JMeter.
N.B.: Le configurazioni dei tool specificati sono presenti nel documento *Test Plan*

4.1 JMH

Per garantire tracciabilità e maggiorare la documentazione relativa al lato tecnico dell'utilizzo della piattaforma, i seguenti dati saranno disponibili nella directory dei risultati nel package di JMH all'interno della repository di UniClass. I dati sono i seguenti:

Categoria	Throughput (ops/us)	Avg Time (us/op)	Sample Time (us/op)
Anno Didattico	0.223	4.503	5.539
Aula	0.223	4.534	5.838
Coordinatore	821.643	0.002	0.027
Corso Laurea	0.228	4.429	5.569
Docente	692.139	0.002	0.028
Lezione	0.217	4.635	5.371
Messaggio	0.221	4.522	6.546
Personale TA	494.982	0.002	0.027
Resto	0.220	4.545	5.488
Topic	0.220	4.566	6.165
Utente	181.690	0.006	0.031

Tabella 3: Sintesi Risultati Benchmark JMH per Categoria (Baseline Pre-Refactoring)

Benchmark	Throughput (ops/us)	Avg Time (us/op)	Sample Time (us/op)
`LoginAccademico`	141.27 ± 0.19	0.00709 ± 0.00001	0.032 ± 0.002
`LoginFallito`	139.31 ± 0.15	0.00720 ± 0.00003	0.032 ± 0.002
`LoginPersonaleTA`	264.49 ± 0.57	0.00378 ± 0.00001	0.028 ± 0.001

Tabella 4: Dettaglio Benchmark JMH - Categoria Utente (Baseline)

Benchmark	Throughput (ops/us)	Avg Time (us/op)	Sample Time (us/op)
`TrovaPerCorso`	298.17 ± 3.04	0.003 ± 0.000	0.029 ± 0.004
`TrovaPerCorsoFallito`	295.39 ± 2.88	0.003 ± 0.000	0.030 ± 0.007
`TrovaPerMatricola`	1084.34 ± 4.44	0.001 ± 0.000	0.025 ± 0.000
`TrovaPerMatricolaFallito`	1090.66 ± 4.20	0.001 ± 0.000	0.026 ± 0.001

Tabella 5: Dettaglio Benchmark JMH - Categoria Docente (Baseline)

Benchmark	Throughput (ops/us)	Avg Time (us/op)	Sample Time (us/op)
`aggiungiMessaggio`	0.22 ± 0.01	4.558 ± 0.161	6.458 ± 2.500
`rimuoviMessaggio`	0.22 ± 0.01	4.486 ± 0.103	6.184 ± 2.432
`trovaMessaggio`	0.22 ± 0.01	4.583 ± 0.110	7.371 ± 2.938
`trovaTutti`	0.22 ± 0.01	4.460 ± 0.186	6.173 ± 2.401

Tabella 6: Dettaglio Benchmark JMH - Categoria Messaggio (Baseline)

Benchmark	Throughput (ops/us)	Avg Time (us/op)	Sample Time (us/op)
`aggiungiTopic`	0.23 ± 0.00	4.511 ± 0.037	6.559 ± 2.595
`rimuoviTopic`	0.22 ± 0.00	4.542 ± 0.107	5.710 ± 2.111
`trovaCorso`	0.22 ± 0.01	4.575 ± 0.141	5.944 ± 2.262
`trovaCorsoLaurea`	0.22 ± 0.01	4.602 ± 0.152	6.547 ± 2.568
`trovaId`	0.22 ± 0.01	4.638 ± 0.044	6.130 ± 2.337
`trovaNome`	0.22 ± 0.01	4.578 ± 0.112	6.215 ± 2.419
`trovaTutti`	0.22 ± 0.01	4.519 ± 0.165	6.049 ± 2.352

Tabella 7: Dettaglio Benchmark JMH - Categoria Topic (Baseline)

I dati qui riportati, già discussi abbondantemente durante la stesura della prima versione del Test Plan aggiornato per motivi di richiesta di cambiamento dello stato architetturale della piattaforma selezionata, mostrano un elevato fenomeno di *collo di bottiglia*, **Bottleneck**, nel modulo utenti, rendendo Docente la classe maggiormente esposta a vulnerabilità di sicurezza (safety) e maggior numero di latenza, nonchè minor numero di operazioni, per via della gerarchia architetturale inefficace e complessa. Il seguente modulo, essendo parte fondamentale della piattaforma, rende tutti gli altri moduli difficilmente misurabili o, semplicemente, più complessi, come il modulo della messaggistica.

4.2 JMeter

Per la visione sistemica della piattaforma presa in considerazione, è stato necessario l'utilizzo di JMeter per specificare determinati Test Plan da condurre su gruppi di thread presenti su determinate richieste alla piattaforma, in base al Rationale e all'obiettivo specificato nel Test Plan. Più precisamente, si è richiesto il punto di rottura quantitativo della piattaforma, resilienza, stabilità ed affidabilità, per imporre la soddisfacibilità dei goal di Software Dependability. Anche per questa metodologia di Testing si è deciso di creare 2 package differenti per contenere i risultati, stavolta in formato csv, per le metriche dei medesimi test.

Endpoint	Samples	Avg (ms)	Min (ms)	Max (ms)	Error %	Throughput (req/s)
Homepage	1000	13	3	86	0.000%	18.89
Chat	1000	3	0	52	0.000%	18.89
TOTAL	**2000**	**8**	**0**	**86**	**0.000%**	**37.56**

Tabella 8: Risultati JMeter - Load Test

Load Testing Results La media è di 13 ms ed un picco di 86 ms, tempi ottimali per una pagina con contenuti dinamici, così come la latenza del ChatBot ed un throughput accettabile.

Endpoint	Samples	Avg (ms)	Min (ms)	Max (ms)	Error %	Throughput (req/s)
Homepage	16401	13949	4	32166	0.000%	48.79
ChatBot	14224	11349	1	26912	0.000%	42.37
Mappa	12996	7932	0	26888	0.000%	38.73
TOTAL	**43621**	**11308**	**0**	**32166**	**0.000%**	**129.75**

Tabella 9: Risultati JMeter - Spike Test

Spike Testing Results La Homepage ha una degradazione prestazionale molto pesante per via dello startup. I picchi raggiungono troppi secondi (32), inaccettabili. Il throughput è ottimo, nonostante l'insieme delle utenze attive, ma con latenze molto alte che compensano.

Endpoint	Samples	Avg (ms)	Min (ms)	Max (ms)	Error %	Throughput (req/s)
Login Page (GET)	18871	182	1	2795	0.000%	314.36
Login POST (Valid)	18784	595	2	3461	0.000%	312.23
Login POST (Invalid)	18565	599	6	3492	0.000%	309.21
TOTAL	**56220**	**458**	**1**	**3492**	**0.000%**	**933.67**

Tabella 10: Risultati JMeter - Authentication Spike Test

Authentication Spike Testing Results La latenza POST Login è molto alta rispetto alla medesima di metodo d'unità visionata nei Microbenchmark; la Latenza GET è ottimale; c'è un collo di bottiglia nella validazione delle credenziali, più presente nella query che nella validazioen logica.

Endpoint	Samples	Avg (ms)	Min (ms)	Max (ms)	Error %	Throughput (req/s)
Homepage	16630	4580	2	16196	0.000%	45.51
ChatBot	16171	649	0	9485	0.000%	44.31
Mappa	16130	341	0	10566	0.000%	45.26
TOTAL	**48931**	**1883**	**0**	**16196**	**0.000%**	**133.91**

Tabella 11: Risultati JMeter - Stress Test

Stress Testing Results La degradazione è anche qui significativa per la latenza Homepage. Il sistema riesce ad adattarsi meglio rispetto allo spike improvviso, avendo latenze decisamente inferiori. Il throughput è ben distribuito e con 500 utenti il sistema degrada ma non collassa. Si può determinare che la capacità massima del sistema (vera e propria) è maggore di 500 utenti.

5 Strategia di Adeguamento della Test Suite

L'intervento di manutenzione non si è limitato alla sola modifica del codice di produzione, ma ha richiesto una profonda revisione dell'approccio al testing.

5.1 Criteri di Selezione e Strategia di Regressione

La selezione dei casi di test (Test Selection) per la validazione dell'intervento di manutenzione non è stata esaustiva indiscriminata, ma ha seguito la strategia di **Regression Test Selection (RTS)** descritta nel SWEBOK v4 (Sez. 5.1.7). L'obiettivo è stato massimizzare la probabilità di rilevazione dei difetti minimizzando il set di test da eseguire, basandosi sull'Analisi di Impatto (v1.0).

I criteri di inclusione ed esclusione adottati sono stati i seguenti:

5.1.1 1. Criterio basato sull'Analisi di Impatto (Modification Traversal)

In accordo con il *Candidate Impact Set (CIS)* definito nel documento di Impact Analysis, sono stati selezionati per l'esecuzione prioritaria:

- **Directly Affected Tests:** Tutti i test unitari che insistono sulle classi oggetto di refactoring strutturale (`Utente`, `Accademico`, `UserDirectory`). Per questi componenti, la strategia è stata di tipo *White-Box*, riscrivendo i test per coprire i nuovi percorsi di controllo generati dalla logica a Composizione.
- **Ripple Effect Tests:** Sono stati inclusi i test del modulo `Orari` (in particolare `LezioneService` e `CorsoService`) poiché, pur non essendo stati modificati nel codice sorgente, dipendono dalle interfacce fornite dal modulo Utenti.

5.1.2 2. Criterio di Obsolescenza (Test Exclusion)

L'attività di manutenzione ha comportato la rimozione fisica di alcune classi (Refactoring *Replace Inheritance with Delegation*). Di conseguenza, la suite è stata epurata secondo il criterio:

> *"Se un test case verifica un'unità strutturale (es. `StudenteDAO`) non più esistente nel design 'To-Be', tale test deve essere rimosso o migrato verso la nuova unità responsabile (`UtenteDAO/RoleManager`)."*

Questo ha portato alla rimozione di circa 450 test ridondanti che verificavano, tramite ereditarietà, logiche ora centralizzate.

5.1.3 3. Criterio di Copertura Strutturale (Adequacy)

Per i nuovi componenti sviluppati (`UserDirectory`), la selezione dei test è stata guidata dall'obiettivo di soddisfare i seguenti criteri di adeguatezza, misurati grazie al plugin JaCoCo Code Coverage (SWEBOK 5.3.1):

- **Statement Coverage:** 100% delle istruzioni eseguibili.
- **Branch Coverage:** Verifica di tutti i rami decisionali (es. `if (utente.hasRole(...))`).
- **Exception Coverage:** Verifica esplicita di tutti i flussi di errore (es. `AlreadyExistentUserExcep`

5.1.4 Matrice di Tracciabilità della Selezione

La Tabella 12 riassume la strategia applicata per ogni modulo.

Modulo	Strategia RTS	Razionale (SWEBOK/ISO 29119)
Utenti (Core)	*Retest-All (New)*	Modulo riscritto. Necessaria copertura White-Box completa per validare la nuova architettura a ruoli.
Orari	*Regression Safety Net*	Modulo dipendente. Esecuzione Black-Box per garantire assenza di regressioni (Ripple Effect).
Conversazioni	*Selective Retest*	Modulo parzialmente impattato. Testati solo i flussi di invio messaggio che coinvolgono il controllo ruoli.

Tabella 12: Matrice di Selezione dei Test in base all'Impatto

5.2 Refactoring delle Dipendenze nei Test Unitari

Nel contesto dello sviluppo dell'applicazione *UniClass*, il processo di refactoring dei test unitari ha avuto come obiettivo principale il miglioramento della gestione delle dipendenze e l'incremento della robustezza dei test. Durante la fase preliminare, è emerso che diversi test esistenti - a seguito della manutenzione - facevano affidamento su oggetti reali, oppure su istanze non completamente inizializzate, generando comportamenti non deterministici.

Per affrontare questa criticità, si è scelto di introdurre un approccio basato sull'iniezione delle dipendenze mediante l'utilizzo di mock. In particolare, oggetti quali `HttpServletRequest`, `HttpServletResponse` e servizi esterni sono stati sostituiti con istanze simulate fornite da *Mockito*. Il refactoring ha previsto l'introduzione di un metodo di utilità per l'iniezione dei mock nei campi privati delle classi, sfruttando la **Java Reflection**. Tale approccio ha permesso di mantenere l'incapsulamento originale delle classi senza modificare la loro visibilità.

5.3 Strategia di Testing per il Modulo *Utente*

L'attività di testing del modulo *Utente* è stata condotta all'interno del processo di revisione architetturale. Il refactoring preliminare ha avuto un ruolo essenziale nel rendere il codice testabile: l'introduzione di costruttori dedicati e l'eliminazione dei container EJB a favore di POJO testabili ha permesso l'applicazione di un paradigma di *white-box testing*.

La progettazione degli unit test ha seguito un criterio di granularità fine. Nei componenti DAO, l'attenzione si è concentrata sulla corretta gestione delle interazioni con l'*EntityManager*; nei servizi applicativi, l'obiettivo è stato verificare la logica di orchestrazione e la propagazione delle eccezioni (es. `AlreadyExistentUserException`). L'intero processo è stato guidato dall'obiettivo di ottenere la massima copertura (Statement e Branch Coverage) garantendo che ogni possibile percorso eseguibile fosse verificato.

5.4 Dettagli Operativi dell'Implementazione

Al fine di rendere operativa la strategia sopra descritta, sono state implementate soluzioni tecniche specifiche:

5.4.1 Meccanismo di Injection via Reflection

Per testare le classi legacy con campi annotati `@Inject` o `@EJB`, è stato implementato un helper method nella classe base di test che:

1. Accede ai campi privati della classe sotto test (SUT).
2. Forza l'accessibilità (`field.setAccessible(true)`).
3. Inietta il Mock di Mockito, trasformando test di integrazione instabili in Unit Test puri.

5.4.2 Setup e Teardown

Per garantire il principio F.I.R.S.T. (Fast, Independent, Repeatable, Self-Validating, Timely), l'ambiente viene resettato ad ogni esecuzione:

- **@BeforeEach:** Inizializzazione dei Mock (`MockitoAnnotations.openMocks(this)`).
- **@AfterEach:** Chiusura delle risorse per prevenire memory leak.

5.5 Criteri di Accettazione per RNF

Poichè l'intervento di refactoring introduce strutturalmente un livello di indirezione, rispetto alla gerarchia complessa definita in precedenza, la strategia di accettazione si basa su soglie di tolleranza esplicite, come l'overhead massimo del 10% ed un sostenimento del carico non superiore al 10% in meno rispetto al Throughput originario.

6 Risultati Post-Intervento (Nuova Suite) - Funzionale

Di seguito si riportano i dati estratti dal report Surefire relativo alla nuova test suite rifattorizzata.

6.1 Sintesi Quantitativa Nuova Suite

Metrica	Valore	Note
Totale Test Eseguiti	527	Suite ottimizzata e deduplicata
Test Superati (Pass)	527	100% Success Rate
Fallimenti (Failures)	0	Nessuna regressione funzionale
Tempo Totale Esecuzione	13.61 s	Incremento dovuto a mocking intensivo

Tabella 13: Riepilogo Esecuzione Nuova Suite (Fonte: surefire_NewTestSuite.html)

6.2 Dettaglio per Pacchetto (Hotspots)

La distribuzione dei test riflette ora le aree critiche oggetto di refactoring:

- **Orari Service (`it.unisa.uniclass.testing.unit.orari.service`):** 107 test.
- **Orari DAO (`it.unisa.uniclass.testing.unit.orari.service.dao`):** 114 test. Copertura intensiva della persistenza.
- **Utenti Service & DAO:**
  - `UserDirectoryTest`: 13 test (nuovo componente di orchestrazione).
  - `UtenteServiceTest`: 10 test (logica business core).
  - `UtenteDAOTest` e `AccademicoDAOTest`: 13 test totali.
- **Conversazioni Model & Service:** 55 test totali, verificano la messaggistica integrata con i nuovi utenti.

7 Risultati Non-Funzionali Post-Refactoring

A seguito delle modifiche effettuati nei test e nella risoluzione degli incidenti (descritti in seguito) nella compilazione dei Benchmark, è stato possibile rieseguire la suite JMH e JMeter sulla nuova architettura. **N.B.**: I benchmark delle classi, ormai Legacy, sono stati sostituiti con Benchmark Utente/Accademico simulando la creazione di un oggetto di tipo Legacy per non perdere la simulazione energetica dell'istanziazione virtuale della gerarchia utenti definita in precedenza

7.1 JMH

Entità	Benchmark (Operazione)	Avg Time (ms)	Throughput (ops/ms)	Note
Utente (New)	`GetAccademiciPerRuolo`	**4.48**	**0.22**	Miglior Performance
Utente (New)	`Login`	4.62	0.21	Login ottimizzato
Coordinatore	`GetAccademiciPerRuolo`	5.35	0.18	+19% tempo vs Utente
Docente	`GetAccademiciPerRuolo`	5.33	0.19	+18% tempo vs Utente
PersonaleTA	`LoginFallito`	13.99	0.07	Anomalia (Outlier)

Tabella 14: Confronto Performance Reali: Architettura a Ruoli (Utente) vs Legacy

I benchmark confermano che la nuova architettura è più performante dal punto di vista unitario, maggiorando il Throughput e diminuendo la latenza. L'anomalia è nel Login-Fallito, per via della gestione differente delle eccezioni rispetto alla classe Legacy. Grazie alle Query JPA ormai ottimizzate, l'architettura non è semplicemente più manutenibile,

raggiungendo uno dei maggiori motivi della Change Request presentata, bensì anche più efficiente. Per avere una visione energetica unitaria maggiore, però, è necessario visionare i risultati di tutti i package. In seguito vi sono tutti i package dettagliati con conclusione energetica unitaria.

Entity	Benchmark (Operation)	Avg Time (us/op)	Throughput (ops/us)
AnnoDidattico	`trovaTutti`	4451.24	0.000229
AnnoDidattico	`aggiungiAnno`	4547.75	0.000219
Aula	`trovaTutte`	4369.32	0.000224
Aula	`trovaAulaNome`	4580.65	0.000222
Corso	`trovaTutti`	4322.68	0.000231
Corso	`trovaCorsiCorsoLaurea`	4463.94	0.000228
CorsoLaurea	`trovaTutti`	4408.56	0.000231
CorsoLaurea	`rimuoviCorsoLaurea`	4561.03	0.000228
Lezione	`trovaTutte`	4515.64	0.000225
Lezione	`trovaLezioniDocente`	4611.27	0.000221
Lezione	`trovaLezioniOreGiorno`	4744.52	0.000217
Resto	`trovaRestoId`	4605.81	0.000220
Resto	`trovaRestiCorsoLaureaObj`	4708.07	0.000217

Tabella 15: Benchmark Modulo Orari: Latenza e Throughput per Entità

Entity	Benchmark (Operation)	Avg Time (us/op)	Throughput (ops/us)
Messaggio	`trovaTutti`	4.47	0.223
Messaggio	`aggiungiMessaggio`	4.50	0.216
Messaggio	`rimuoviMessaggio`	4.51	0.218
Topic	`trovaTutti`	4.51	0.226
Topic	`aggiungiTopic`	4.58	0.218
Topic	`trovaCorso`	4.58	0.220

Tabella 16: Benchmark Modulo Conversazioni: Analisi CRUD

L'analisi trasversali dei micro-benchmark JMH, condotta sui tre moduli principali e unici, permette di trarre conclusioni sistemiche sull'impatto del refactoring. In particolare, il modulo utenti mostra i miglioramenti più marcati data la minore complessità della gerarchia architetturale. Questo valida la tesi che l'eliminazione dell'ereditarietà complessa ottimizza l'accesso ai dati. Pur non essendo stati riscritti strutturalmente, i moduli orari e conversazioni beneficiano indirettamente dell'entità Utente e della gestione transazionale definita nella sua architettura. Le operazioni di correlazione mantengono performance eccellenti, dimostrando assenza di regressioni da **Ripple Effects**. In conclusione, il sistema non solo rispetta i vincoli di non-regressione, ma offre una base prestazionale superiore, garantendo tempi di risposta unitari mediamente inferiori, correggendo problematiche di misurazione nel sistema Legcacy, per via dei suoi comportamenti inattesi.

7.2 JMeter

I microbenchmark definiscono le unità precise dei package definiti nella repository ma, per il soddisfacimento dei nostri obiettivi, stabiliti nella Change Request e ripetuti nella totalità dei documenti, non bastano. È necessario, di conseguenza, riportare i risultati del Performance Testing system-level con JMeter, strumento utilizzato anche in precedenza.

7.2.1 Risultati Load Test

Endpoint	Samples	Avg (ms)	Max (ms)	Error %	Throughput
GET Homepage	1000	1	28	0.00%	3.09 req/s
GET Chat	1000	0	125	0.00%	3.09 req/s
TOTAL	**2000**	**1**	**125**	**0.00%**	**6.17 req/s**

Tabella 17: JMeter New: Load Test Results

7.2.2 Risultati Authentication Spike Test

Label	Samples	Avg (ms)	Max (ms)	Err %	Throughput
GET Login Page	165474	45	252	0.00%	507.45 req/s
POST Login (Valid)	165188	234	1274	0.00%	506.33 req/s
POST Login (Invalid)	164364	268	1372	0.00%	503.90 req/s
TOTAL	**495026**	**182**	**1372**	**0.00%**	**1517.10 req/s**

Tabella 18: JMeter New: Authentication Spike Results

7.2.3 Risultati General Spike Test

Label	Samples	Avg (ms)	Max (ms)	Err %	Throughput
GET Homepage	312123	215	20193	0.00%	2543.46 req/s
GET ChatBot	311131	113	18664	0.00%	2537.50 req/s
GET Mappa	310698	115	19042	0.00%	2536.10 req/s
TOTAL	**933952**	**148**	**20193**	**0.00%**	**7610.62 req/s**

Tabella 19: JMeter New: General Spike Results

7.2.4 Risultati Stress Test

Label	Samples	Avg (ms)	Max (ms)	Err %	Throughput
GET Homepage	20272	1039	118012	3.15%	54.14 req/s
GET ChatBot	20175	745	103360	3.01%	53.88 req/s
GET Mappa	20062	785	104107	2.82%	53.63 req/s
TOTAL	**60509**	**857**	**118012**	**2.99%**	**161.48 req/s**

Tabella 20: JMeter New: Stress Test Results

Il confronto tra le esecuzioni Pre-Refactoring e Post-Refactoring evidenzia il miglioramento radicale delle prestazioni, soprattutto sulla gestione dei picchi di carico.

Load Test : Si ha un leggero decremento nel Throughput e latenza. Questa differenza, però, è l'unico punto discretamente negativo del cambiamento architetturale, dati i seguenti tipi di test soddisfatti.

General Spike La nuova architettura ha dimostrato una capacità di assorbimento del traffico molto più alta, un incremento di quasi 60x che suggerisce come la rimozione delle query polimorfiche complesse abbia sbloccato la concorrenza. La latenza media, invece, è crollata di 11s, smaltendo perfettamente le richieste in tempo reale.

Authentication Spike Conferma la bontà del refactoring. Infatti la latenza media diminuisce del 60%, grazie alla semplificazione delle query di verifica delle credenziali, determinando un'importante differenza tra autenticazione ed autorizzazione. Il throughput, inoltre, è aumentato di 1384 richieste al secondo.

Stress Test La versione to-be si è gestito un campione superiore, mantenendo una latenza media inferiore di più del 55% rispetto alla baseline. Si nota la comparsa di un lieve tasso di errore, *assente nella baseline*, naturale dato il throughput molto più elevato, "spingendo" l'hardware molto di più, arrivando più velocemente (maggiorando anche le richieste) ai limiti fisici delle connessioni TCP o dei thread disponibili dal gruppo del Test Plan JMeter.

Lato tecnico JPQL Non ci siamo fermati alla comprensione dei valori definiti dai nuovi test del Test Plan JMeter: abbiamo voluto analizzare le motivazioni dal punto di vista della base di dati sull'inefficienza di una gerarchia e architettura polimorfica. La motivazione principale si chiama **Table Lock** o **Row Lock** quando, per via di un'efficienza dal punto di vista strutturale nelle Join del Database, si *chiude* erroneamente la possibilità di modificare o richiedere informazioni da delle tabelle. Motivo per cui, soprattutto in JPQL, in un livello molto più alto rispetto alla base di dati a cui è connessa l'applicazione, si possono avere problematiche dal punto di vista di Throughput e Latenza anche nelle semplici query di retrieval.

8 Valutazione Comparativa e Conclusioni

8.1 Valutazione e Conformità (Evaluation)

L'attività di testing è considerata conclusa e valida solo al soddisfacimento dei criteri di adeguatezza (*Test Adequacy Criteria*) definiti nel Piano di Test.

8.2 Verifica dei Criteri di Adeguatezza

La Tabella 21 confronta i risultati ottenuti con le soglie di accettazione stabilite.

Criterio di Adeguatezza	Soglia Minima	Stato Corrente
Pass Rate (Percentuale Successo)	100%	**100%** (Soddisfatto)
Defect Density (Bug aperti)	0 Bloccanti	**0** (Soddisfatto)
Regression Stability	Nessuna regressione	**Confermata**
Statement Coverage (Modulo Utenti)	$> 80\%$	**High** (Garantito da White-Box)
Performance Degradation	$< 10\%$	**Nessuna** (Miglioramento)
Scalabilità (Spike Test)	No Crash	**Soddisfatto** (Max 7610 req/s)

Tabella 21: Conformità ai Criteri di Adeguatezza

8.3 Valutazione Comparativa Pre vs Post

8.4 Confronto Pre vs Post Refactoring

La Tabella 22 evidenzia l'evoluzione della strategia di testing.

Indicatore	Baseline (As-Is)	Nuova Suite (To-Be)	Delta (Δ)
Test Count	969	527	-442
Success Rate	100%	100%	=
Tempo Esecuzione	7.199 s	13.610 s	+6.41 s
Approccio Utenti	Ereditarietà	UserDirectory	Refactoring
Latenza Media	~4.6 us	~4.5 us	-2% (Stabile)
Max Throughput (Spike)	129 req/s	7610 req/s	**+5800%**

Tabella 22: Analisi Comparativa delle Performance di Test

8.5 Analisi delle Variazioni

Le variazioni metriche sono giustificate dalle seguenti scelte progettuali:

1. **Riduzione Test Count (-45%):** La baseline conteneva numerosi test ridondanti generati per le sottoclassi (`Studente`, `Docente`) che testavano la stessa logica ereditata. Con il passaggio alla *Composizione*, la logica è centralizzata in `Utente` e `UserDirectory`, eliminando la necessità di duplicare i test per ogni ruolo.

2. **Incremento Tempo di Esecuzione:** L'aumento del tempo totale (da 7s a 13s) è dovuto all'introduzione sistematica di *Mockito*. Sebbene l'inizializzazione dei mock comporti un leggero overhead (overhead di reflection e proxying), essa garantisce un isolamento reale che la suite precedente (spesso basata su stati inconsistenti) non possedeva.

3. **Incremento del Throughput (+5800%):** Il passaggio da una gerarchia di classi (Joined Inheritance) a una tabella singola centralizzata (`Utente`) ha eliminato le

complesse operazioni di JOIN che bloccavano il database durante i picchi di carico. Questo ha sbloccato la scalabilità verticale del sistema, permettendo di gestire migliaia di richieste concorrenti senza saturare le risorse.

8.6 Esito Finale

Il refactoring del modulo Utenti (Change Request v1.0) è stato validato con successo.

- **Integrità:** I test del modulo *Orari* (il più popoloso con 200+ test DAO/Service) passano correttamente, dimostrando assenza di *Ripple Effect* negativi.
- **Manutenibilità:** La nuova suite è più concisa, priva di duplicazioni e semanticamente allineata alla nuova architettura.
- **Prestazioni:** I benchmark dimostrano che il sistema non solo non ha subito regressioni, ma ha acquisito una capacità di carico di ordine superiore, risolvendo colli di bottiglia architetturali.

Esito Finale: Il nuovo sistema è **accettato** secondo il superamento delle soglie definite in precedenza.

9 Mutation Testing Assessment

A completamento delle attività di verifica e validazione della nuova test suite, è stato condotto un processo di Mutation Testing mediante il tool PIT (versione 1.16.3), con l'obiettivo di valutare la capacità dei test di individuare difetti introdotti artificialmente nel codice sorgente. Tale tecnica, coerente con i criteri avanzati di adeguatezza descritti nello standard ISO/IEC/IEEE 29119-4, consente di misurare la robustezza effettiva dei test oltre la semplice copertura strutturale.

L'analisi è stata eseguita sulla versione post-refactoring del sistema, in ambiente coerente con quello utilizzato per l'esecuzione della test suite.

9.1 Configurazione e Perimetro di Analisi

L'esecuzione è stata configurata per includere esclusivamente i test di unità, escludendo i test di integrazione localizzati nel package `it.unisa.uniclass.testing.integration`, poiché tali test dipendono da componenti esterni (Selenium, Testcontainers, Arquillian) non compatibili con l'ambiente isolato richiesto da PIT.

Il perimetro di mutazione ha incluso tutte le classi applicative del progetto:

- `targetClasses`: `it.unisa.uniclass.*`
- `targetTests`: `it.unisa.uniclass.testing.unit.*`

La configurazione ha utilizzato il set di mutatori `DEFAULTS`, che comprende alterazioni su operatori logici, boundary conditions, valori di ritorno, rimozione di chiamate a metodi void e manipolazioni dei rami condizionali.

9.2 Risultati Quantitativi

L'esecuzione ha prodotto i seguenti risultati complessivi:

- **Line Coverage**: 92% (1132/1228 linee mutate coperte)
- **Mutazioni Generate**: 492
- **Mutazioni Uccise**: 378 (**77%**)
- **Mutazioni Sopravvissute**: 114
- **Mutazioni Senza Copertura**: 30
- **Test Strength**: 82%
- **Test Eseguiti**: 653 (1.33 test per mutazione)

Il mutation score del 77% indica una buona capacità della suite di rilevare alterazioni semantiche, risultando in linea con valori comunemente considerati adeguati in contesti enterprise basati su architetture servlet, DAO e servizi applicativi.

9.3 Analisi per Pacchetto

Il report evidenzia differenze significative tra i vari moduli:

- **Modulo Orari**: mutation coverage tra 64% e 100%, con performance eccellenti nei DAO (100%) e nei model (97%).
- **Modulo Utenti**: valori compresi tra 76% e 96%, con ottima capacità di rilevazione nei componenti core (UserDirectory, UtenteService).
- **Modulo Conversazioni**: mutation coverage tra 58% e 82%, con sopravvivenze concentrate nei controller, principalmente su rami di errore.
- **Modulo Common.Security**: coverage più basso (20%), dovuto alla presenza di controlli difensivi e rami non osservabili dai test.

9.4 Interpretazione dei Mutanti Sopravvissuti

L'analisi qualitativa dei mutanti sopravvissuti mostra che essi si concentrano in tre categorie principali:

1. **Rami di errore non verificati**: in particolare nei controller, dove condizioni come `else` o `error branch` non sono coperte da test specifici.
2. **Metodi void con side-effect non osservabili**: il mutatore `VoidMethodCallMutator` ha generato 195 mutazioni, di cui 37 sopravvissute, tipicamente in metodi che aggiornano lo stato interno senza restituire valori.
3. **Boundary conditions**: mutazioni su operatori logici (es. `>=` in `>`) non sempre rilevate, indicando la necessità di test più mirati sui casi limite.

Questi risultati non indicano difetti funzionali del sistema sotto test, ma suggeriscono possibili miglioramenti nella granularità dei test, in particolare nei controller e nei flussi di errore.

9.5 Conclusioni del Mutation Testing

Il Mutation Testing conferma che la nuova test suite:

- è robusta e in grado di rilevare la maggior parte delle alterazioni semantiche;
- presenta un'elevata copertura strutturale e mutazionale;
- è coerente con i criteri di adeguatezza definiti nel Test Plan;
- non mostra regressioni nei moduli rifattorizzati.

Il mutation score del 77% e il test strength dell'82% rappresentano indicatori di qualità complessivamente positivi per il sistema UniClass post-refactoring.

Permane tuttavia un margine di miglioramento nei rami di errore dei controller e nei metodi void con side-effect, che rappresentano le principali aree di rischio residuo.

10 Incident Summary

10.1 Incident Overview

Le anomalie osservate sono state classificate come incidenti di regressione dovuti a disallineamento tra la suite di test legacy e la nuova architettura applicativa.

Durante l'esecuzione della suite di test automatizzati del progetto *UniClass*, il processo di build ha evidenziato una regressione critica con un numero significativo di errori distribuiti su diversi moduli. L'incidente è emerso durante un'esecuzione completa del comando `mvn clean package`. Le failure non sono riconducibili a bug funzionali, ma a un disallineamento tra il codice di test pre-esistente e la nuova architettura del sistema basata sulla composizione e sull'iniezione delle dipendenze tramite Facade.

10.2 Panoramica del problema

I log di Maven e Surefire hanno riportato una combinazione di:

- `Compilation Errors`: Impossibilità di compilare i benchmark JMH a causa di modifiche ai costruttori dei Service.
- `IndexOutOfBoundsException`: Fallimento dei test basati su `MockedConstruction`.
- `NullPointerException` (NPE): Mancata inizializzazione dei campi annotati con `@EJB`.
- `PotentialStubbingProblem`: Mismatch degli argomenti durante il mocking dei metodi `merge` e `persist`.
- `Assertion Failures`: Discrepanze tra il comportamento atteso delle servlet e l'effettivo protocollo di errore HTTP utilizzato.

10.3 Test Case maggiormente impattati

I seguenti test hanno evidenziato failure sistematiche:

- `LoginServletTest`
- `TopicServiceTest`
- `GetEmailServletTest`
- `AttivaUtentiServletTest`
- `LezioneDAOTest`

10.4 Errori principali osservati

1. Errori di Compilazione Benchmark (JMH) I moduli di benchmark cercavano di istanziare classi eliminate (*Docente*, *PersonaleTA*) o Service tramite costruttori con parametri non più presenti. Questo ha impedito la generazione dei dati sulle performance post-refactoring.

2. NPE e Fallimento dell'Iniezione Dipendenze Poiché JUnit opera al di fuori del container Jakarta EE, i campi privati marcati con `@EJB` risultavano `null`. Questo ha causato NPE sistematiche in test cruciali come `LoginServletTest` e `TopicServiceTest`, poiché il flusso logico tentava di invocare metodi su oggetti mai inizializzati.

3. Incoerenza nei protocolli di errore HTTP In test come `GetEmailServletTest` e `AttivaUtentiServletTest`, le asserzioni cercavano una specifica azione (`sendRedirect`), mentre il controller eseguiva `sendError(500)` o `setStatus(500)`. Tale discrepanza ha evidenziato una mancanza di standardizzazione nella gestione globale delle eccezioni tra i diversi controller.

4. IndexOutOfBoundsException (`MockedConstruction`) L'uso della tecnica `MockedConstructio` è risultato inefficace: intercettando solo le istanze create tramite operatore `new`, la lista `constructed()` risultava vuota nelle servlet che ora delegano l'istanziazione al container. L'accesso all'indice 0 della lista ha quindi generato eccezioni fatali.

5. PotentialStubbingProblem (DAO Mismatch) In `LezioneDAOTest`, Mockito ha rilevato stubbing non corrispondenti: il codice sotto test invocava l'EntityManager con istanze diverse da quelle pre-configurate nel test, segnalando una fragilità nei test di unità che non tengono conto del cambio di stato (Attached/Detached) degli oggetti persistenti.

10.5 Analisi delle cause radice

L'incidente è riconducibile a due fattori principali:

1. **Evoluzione del Modello di Dominio:** La transizione dall'eredità alla composizione (*Accademico* + *Ruolo*) ha rotto i contratti semantici dei test pre-esistenti.

2. **Disallineamento Strutturale:** L'ambiente di test JUnit non fornisce un container per l'iniezione automatica. L'uso di vecchie tecniche di mocking (`MockedConstruction`) è diventato incompatibile con la nuova strategia basata su *UserDirectory* (Facade Pattern).

10.6 Impatto

Il processo di build è stato interrotto, rendendo impossibile:

- Verificare la correttezza del modulo *conversazioni* post-refactoring.
- Generare un report di *Code Coverage* affidabile tramite JaCoCo.
- Eseguire i test di carico e benchmark previsti per il modulo *orari*.

10.7 Risoluzione e Conclusione

La suite di test ha richiesto un intervento correttivo basato su:

- Sostituzione integrale di `MockedConstruction` con iniezione manuale via **Reflection** (per simulare il comportamento del container su campi privati).
- Utilizzo di **Wrapper Class** interne ai test per esporre i metodi `protected` delle servlet senza alterarne la visibilità nel codice sorgente.
- Standardizzazione dei test di errore per riflettere l'uso di `sendError(500)` in caso di eccezioni non gestite.
- Aggiornamento dei matcher di Mockito (`any()`) per gestire il merge di entità durante i test dei DAO.

10.8 Stato degli incident

Gli incidenti identificati sono stati classificati come difetti della test suite e non del sistema sotto test. È stato pianificato un intervento di refactoring della suite di test al fine di ristabilire la coerenza con la nuova architettura.

Lo stato corrente degli incidenti è: **Closed – Test Suite Refactoring done**.