



Università degli Studi di Salerno

Corso di Ingegneria del Software: Tecniche Avanzate

Code Smile Post-Modification Testing Document Versione 1.0

Team:

Antonio Caiazza
Emanuele Iovane
Salvatore Di Martino



| | |
|---|------------------|
| Progetto: Code Smile | Versione: 1.0 |
| Documento: Post-Modification Testing Document | Data: 09/01/2026 |

Partecipanti:

| Nome | Matricola |
|----------------------|------------|
| Antonio Caiazzo | NF22500205 |
| Salvatore Di Martino | NF22500114 |
| Emanuele Iovane | NF22500162 |
| | |
| | |
| | |

| | |
|--------------------|--|
| Scritto da: | Antonio Caiazzo, Salvatore Di Martino, Emanuele Iovane |
|--------------------|--|

| | |
|---|------------------|
| Progetto: Code Smile | Versione: 1.0 |
| Documento: Post-Modification Testing Document | Data: 09/01/2026 |

Indice

| | | |
|--------|--|----|
| 1. | Introduzione..... | 4 |
| 1. | Post-Modification Testing | 5 |
| 1.1. | CR01 - Allineamento dinamico degli import e degli endpoint tra ambiente locale e containerizzato | 5 |
| 1.2. | CR02 – Generazione del Call Graph nella CLI | 5 |
| 1.3. | CR03 – Generazione del Call Graph nella WebApp | 6 |
| 2. | Regression Testing | 7 |
| 2.1. | Obiettivo..... | 7 |
| 2.2. | Strategia..... | 7 |
| 2.3. | Esecuzione Test Suite | 7 |
| 2.3.1. | Test automatici | 7 |
| 2.3.2. | Copertura..... | 7 |
| 3. | Dettaglio dei Test..... | 8 |
| 3.1. | Unit Test..... | 8 |
| 3.1.1. | CR02 – Call Graph (CLI)..... | 8 |
| 3.1.2. | CR03 – Call Graph (WebApp)..... | 10 |
| 3.2. | Integration Test | 11 |
| 3.2.1. | CR02 – CLI -> call_graph.json..... | 11 |
| 3.2.2. | CR03 – Gateway → Static Analysis Service (Call Graph)..... | 11 |
| 3.1. | System Test | 12 |
| 3.1. | End-to-End Testing (WebApp) | 13 |
| 4. | Conclusione | 13 |

| | |
|---|------------------|
| Progetto: Code Smile | Versione: 1.0 |
| Documento: Post-Modification Testing Document | Data: 09/01/2026 |

1. Introduzione

Il presente documento descrive le attività di testing svolte sul sistema **CodeSmile** in seguito all'introduzione delle Change Requests **CR01**, **CR02** e **CR03**. Gli obiettivi sono stati:

- **Post-Modification Testing:** verificare il corretto funzionamento delle nuove funzionalità introdotte.
- **Regression Testing:** assicurare che il sistema preesistente non abbia subito regressioni funzionali.

Ambiente di Testing

- **Sistema Operativo:** MacOS Tahoe 26.1
- **Python:** 3.11
- **Node.js:** 18+
- **Dipendenze:** installate tramite i file *requirements.txt* e *package.json*
- **Repository:** https://github.com/Antonio-Caiazzo/smell_ai

| | |
|---|------------------|
| Progetto: Code Smile | Versione: 1.0 |
| Documento: Post-Modification Testing Document | Data: 09/01/2026 |

1. Post-Modification Testing

Le attività di Post-Modification Testing hanno incluso la verifica delle componenti introdotte o modificate dalle CR, mantenendo l'attenzione sui confini funzionali dichiarati (moduli di **GUI** o **AI**). Tutti i test hanno avuto esito positivo, senza rilevazione di anomalie, in conformità con gli obiettivi definiti nel **Master Test Plan**.

1.1.CR01 - Allineamento dinamico degli import e degli endpoint tra ambiente locale e containerizzato

Tipo di modifica: Preventive

Componenti coinvolte:

Sono stati coinvolti (e modificati) **14 file**, includendo gateway, servizi (AI/static/report) e configurazioni Docker, tra cui:

- *webapp/gateway/main.py*
- *webapp/services/aiservice/app/main.py, router/detect_smell.py, utils/model.py*
- *webapp/services/staticanalysis/app/main.py, router/detect_smell.py, utils/static_analysis.py*
- *webapp/services/report/app/main.py, router/report.py*
- *docker-compose.yml* e *Dockerfile* dei servizi

Verifiche effettuate:

- Validazione manuale della configurazione dinamica degli endpoint
- Verifica manuale del build context unificato e della corretta visibilità dei package.

Esito: Funzionalità verificata in ambiente locale e allineamento predisposto per l'esecuzione containerizzata. Nessun problema rilevato durante la verifica manuale per l'ambiente locale e containerizzato.

1.2.CR02 – Generazione del Call Graph nella CLI

Tipo di modifica: Perfective

Componenti coinvolte:

Sono stati coinvolti **3 file**:

- *cli/cli_runner.py,*
- *components/call_graph_generator.py,*
- *components/project_analyzer.py*

Verifiche effettuate:

Con la CR02 sono stati sviluppati nuovi test su tre livelli:

| | |
|---|------------------|
| Progetto: Code Smile | Versione: 1.0 |
| Documento: Post-Modification Testing Document | Data: 09/01/2026 |

- **Unità:** verificano che il *CallGraphGenerator* analizzi correttamente il codice Python per identificare funzioni e metodi come nodi e per costruire correttamente le relazioni di chiamata tra di essi come archi nel call graph.
- **Integrazione:** verificano che l'intera pipeline di generazione del call graph, dall'esecuzione della CLI fino alla produzione del file *call_graph.json*, funzioni correttamente integrando CLI, *Analyzer* e *CallGraphGenerator*.
- **Sistema:** verificano la nuova funzionalità, estendendo il Category Partition con la nuova categoria **CG (Call Graph)** legata ai parametri di configurazione del tool.

Esito: Tutti i test funzionali di unità, integrazione e sistema hanno avuto esito positivo.

1.3.CR03 – Generazione del Call Graph nella WebApp

Tipo di modifica: Perfective

Componenti coinvolte:

Sono stati coinvolti 11 file:

Frontend:

- *webapp/app/call-graph/page.tsx* (nuova pagina),
- *webapp/app/page.tsx*, *webapp/utils/api.ts*,
- *webapp/components/HeaderComponent.tsx*

Backend:

- *webapp/gateway/main.py*,
- *webapp/services/staticanalysis/app/routers/detect_smell.py*,
- *webapp/services/staticanalysis/app/utils/static_analysis.py*,
- *webapp/services/staticanalysis/app/schemas/requests.py*,
- *webapp/services/staticanalysis/app/schemas/responses.py*
- *components/call_graph_generator.py* (estensione di informazioni per supportare informazioni aggiuntive nella webapp)
- *webapp/package.json* (necessità gestione ZIP lato client tramite *jszip*)

Verifiche effettuate:

Con la CR03 sono stati introdotti nuovi livelli di test per coprire sia la logica di generazione (Backend) che l'interattività (Frontend).

- **Backend Tests:**
 - Verifica della corretta estrazione di **nodi** (funzioni, classi) e **archi** (chiamate).
 - Verifica della gestione dei percorsi file (normalizzazione relativa per il matching).
- **Frontend Tests:**
 - **Rendering:** la pagina carica correttamente titolo e descrizione;
 - **Upload:** simulazione upload ZIP e verifica che la chiamata API venga effettuata;
 - **Interazione grafo:** mock ReactFlow per verificare rendering nodi nel DOM;

| | |
|---|------------------|
| Progetto: Code Smile | Versione: 1.0 |
| Documento: Post-Modification Testing Document | Data: 09/01/2026 |

- **Modale sorgente:** click su nodo per verificare che appaia il pop-up con codice sorgente.

Esito: Funzionalità conforme ai requisiti. I test relativi alla funzionalità sono passati.

2. Regression Testing

2.1. Obiettivo

Garantire che le modifiche introdotte dalle CR non abbiano alterato il comportamento del sistema nelle funzionalità preesistenti (CLI/WebApp), mantenendo compatibilità e stabilità.

2.2. Strategia

La regressione è stata condotta con:

- Riesecuzione completa della suite di test preesistenti: unità, integrazione, sistema.
- Confronto degli output generati con quelli attesi.

2.3. Esecuzione Test Suite

2.3.1. Test automatici

È stata eseguita una suite completa di **test di regressione** sull'intera codebase, coprendo componenti **Backend**, **Frontend** e **System**. I test relativi ai **moduli AI** e alla **GUI** sono stati esplicitamente esclusi o ignorati.

Stato complessivo: SUPERATO (con esclusioni previste)

| Categoria di test | Tipo | Totale | Superati | Falliti | Tasso di successo |
|-------------------|------------------|--------|----------|---------|-------------------|
| Unit Tests | Backend (Pytest) | 197 | 197 | 0 | 100% |
| Integration | Backend Logic | 5 | 5 | 0 | 100% |
| Integration | Gateway API | 5 | 5 | 0 | 100% |
| System Tests | Script Completo | 28 | 28 | 0 | 100% |
| Frontend | Unit (Jest) | 29 | 29 | 0 | 100% |
| Frontend | E2E (Cypress) | 23 | 23 | 0 | 100% |
| Totale | | 287 | 287 | 0 | 100% |

2.3.2. Copertura

Branch Coverage: 88%

Il valore di coverage complessivo (linee e rami) è pari a 88%. I moduli core superano il 97% per i rilevatori principali. La copertura parziale è limitata ad alcuni rami condizionali nelle componenti *GUI*, *report* e *utils*.

| | |
|---|------------------|
| Progetto: Code Smile | Versione: 1.0 |
| Documento: Post-Modification Testing Document | Data: 09/01/2026 |

3. Dettaglio dei Test

In questo capitolo vengono riportati i principali casi di test introdotti o aggiornati per ciascuna Change Request, in modo da fornire tracciabilità puntuale tra requisito, componente e verifica eseguita.

3.1. Unit Test

3.1.1. CR02 – Call Graph (CLI)

| ID | Funzione/Classe | Input | Oracolo atteso |
|------------|---|--|---|
| UT-CR02-01 | <i>CallGraphGenerator.generate</i> | Progetto con più file Python con funzioni che si chiamano | Grafo con nodi per definizioni e archi per chiamate risolvibili |
| UT-CR02-02 | <i>CallGraphGenerator.generate_dot</i> | File valido con almeno una funzione | DOT corretto con almeno il nodo rilevato |
| UT-CR02-03 | <i>CallGraphGenerator.generate_dot</i> | File valido; cache già popolata | DOT generato usando dati presenti (senza rescansione) |
| UT-CR02-04 | <i>CallGraphGenerator.generate_dot</i> | Chiamate multiple allo stesso target (stesso source→target ripetuto) | Arco DOT con etichetta conteggio chiamate (>1) |
| UT-CR02-05 | <i>CallGraphGenerator.generate_dot</i> | Singola chiamata a un target | Arco DOT senza label di conteggio |
| UT-CR02-06 | <i>CallGraphGenerator._get_module_info</i> | relpath non calcolabile | Fallback su basename (nessuna eccezione) |
| UT-CR02-07 | <i>CallGraphGenerator._get_line_snippet</i> | File leggibile e riga valida | Restituisce snippet della riga richiesta |
| UT-CR02-08 | <i>CallGraphGenerator._get_line_snippet</i> | File leggibile e riga fuori range | Restituisce stringa vuota |
| UT-CR02-09 | <i>CallGraphGenerator._get_line_snippet</i> | File non leggibile / errore I/O | Restituisce stringa vuota senza crash |
| UT-CR02-10 | <i>CallGraphGenerator._get_source_segment</i> | File leggibile e range righe valido | Restituisce segmento corrispondente |
| UT-CR02-11 | <i>CallGraphGenerator._get_source_segment</i> | File leggibile; range non valido | Restituisce “Source not available” |
| UT-CR02-12 | <i>CallGraphGenerator._get_source_segment</i> | File non leggibile / errore I/O | “Source not available” senza crash |
| UT-CR02-13 | <i>CallGraphGenerator._scan_definitions</i> | File con sintassi non valida | Parsing ignorato in modo sicuro |
| UT-CR02-14 | <i>CallGraphGenerator._scan_calls</i> | File con sintassi non valida | Parsing ignorato in modo sicuro |
| UT-CR02-15 | <i>CallGraphGenerator.generate</i> | Chiamate in global scope (fuori da funzioni/classi) | Chiamate globali ignorate (niente archi “orfani”) |
| UT-CR02-16 | <i>CallGraphGenerator._add_node</i> | Inserimento ripetuto stesso nodo | Idempotente: nodo non duplicato/sovrascritto |
| UT-CR02-17 | <i>CallGraphGenerator._add_edge</i> | Inserimento ripetuto stesso arco | Arco unico, callsites incrementata |

| | |
|---|------------------|
| Progetto: Code Smile | Versione: 1.0 |
| Documento: Post-Modification Testing Document | Data: 09/01/2026 |

| | | | |
|------------|--|--|---|
| UT-CR02-18 | <i>CallGraphGenerator.generate</i> | Funzioni async e chiamate await | Nodi/archi corretti anche per async |
| UT-CR02-19 | <i>CallGraphGenerator.generate</i> | Chiamata a funzione nello stesso modulo | Target risolto correttamente e arco creato |
| UT-CR02-20 | <i>CallGraphGenerator.generate</i> | Due file: uno definisce funzione, altro la usa via import | Risoluzione cross-file e arco creato |
| UT-CR02-21 | <i>CallGraphGenerator (fallback)</i> | Chiamata cross-file senza mapping import | Fallback per suffisso: arco creato se univoco |
| UT-CR02-22 | <i>CallGraphGenerator.generate</i> | Istanziamento classe con <code>__init__</code> | Risoluzione verso <code>__init__</code> e arco creato |
| UT-CR02-23 | <i>CallGraphGenerator.generate</i> | Chiamata non riconducibile a Name/Attribute | Chiamata ignorata (nessun arco) |
| UT-CR02-24 | <i>CallGraphGenerator.generate</i> | Chiamata a metodo con nome univoco nel progetto | Metodo risolto univocamente e arco creato |
| UT-CR02-25 | <i>CallGraphGenerator.generate</i> | Metodo non univoco ma oggetto “riconoscibile” (euristica nome) | Target selezionato via euristica e arco creato |
| UT-CR02-26 | <i>CallGraphGenerator.generate</i> | Metodo su espressione non semplice ma metodo univoco | Target risolto tramite unicità e arco creato |
| UT-CR02-27 | <i>CallGraphGenerator.generate</i> | <code>self.metodo()</code> con metodo nella classe corrente | Risoluzione corretta e arco creato |
| UT-CR02-28 | <i>CallGraphGenerator.generate</i> | Chiamata a funzione inesistente/sconosciuta | Nessun arco generato |
| UT-CR02-29 | <i>CallGraphGenerator.generate</i> | <code>self.metodo()</code> ma metodo non definito nei nodi | Nessun arco (evita falsi positivi) |
| UT-CR02-30 | <i>CallGraphGenerator.generate</i> | Metodo ambiguo su oggetto non identificabile | Nessun arco (evita risoluzioni errate) |
| UT-CR02-31 | <i>CallGraphGenerator.generate</i> | Metodo ambiguo con mismatch euristica oggetto | Nessun arco (nessun match affidabile) |
| UT-CR02-32 | <i>ProjectAnalyzer.generate_call_graph</i> | Progetto con file Python; generator disponibile | Genera e salva output call graph (JSON e DOT) |
| UT-CR02-33 | <i>ProjectAnalyzer._save_results</i> | DataFrame vuoto + nome file | Nessun file creato e nessuna directory necessaria |
| UT-CR02-34 | <i>ProjectAnalyzer.analyze_projects_sequential</i> | Base path non esistente | Base path creato automaticamente |
| UT-CR02-35 | <i>ProjectAnalyzer.analyze_projects_sequential</i> | <code>resume=True</code> con log già presente | Salta progetti \leq “last project”, analizza successivi |
| UT-CR02-36 | <i>ProjectAnalyzer.analyze_projects_sequential</i> | Base path con elementi non-directory | Ignorati, nessun crash |
| UT-CR02-37 | <i>ProjectAnalyzer.analyze_projects_sequential</i> | Errore generico durante analisi progetto | Errore gestito, continua sugli altri |
| UT-CR02-38 | <i>ProjectAnalyzer.analyze_projects_sequential</i> | Progetto con 0 smell | Nessun CSV dettagli creato |
| UT-CR02-39 | <i>ProjectAnalyzer.analyze_projects_parallel</i> | Base path non esistente | Base path creato prima del parallelo |
| UT-CR02-40 | <i>ProjectAnalyzer.analyze_projects_parallel</i> | Base path con directory escluse e/o non-directory | Elementi esclusi ignorati |

| | |
|---|------------------|
| Progetto: Code Smile | Versione: 1.0 |
| Documento: Post-Modification Testing Document | Data: 09/01/2026 |

| | | | |
|------------|--|---|--|
| UT-CR02-41 | <i>ProjectAnalyzer.analyze_projects_parallel</i> | Progetto con smell > 0 | Output informativo coerente ("Found X...") |
| UT-CR02-42 | <i>ProjectAnalyzer._save_results</i> | DataFrame non vuoto + nome file | CSV creato con contenuto coerente |
| UT-CR02-43 | <i>ProjectAnalyzer.analyze_project</i> | Progetto con file validi ma 0 smell | Termina correttamente, ritorna 0 |
| UT-CR02-44 | <i>ProjectAnalyzer.analyze_projects_sequential</i> | Presenza output + execution_log.txt + progetto | Salta output/log e analizza solo progetti reali |
| UT-CR02-45 | <i>ProjectAnalyzer.analyze_projects_sequential</i> | Durante scansione file: SyntaxError/FileNotFoundError | Errore registrato (es. error.txt), continua |
| UT-CR02-46 | <i>ProjectAnalyzer.analyze_projects_parallel</i> | In parallelo con 0 smell | Nessun messaggio "Found X..." |
| UT-CR02-47 | <i>ProjectAnalyzer.analyze_projects_parallel</i> | Errore generico nel task parallelo | Gestito e segnalato senza interrompere globale |
| UT-CR02-48 | <i>CodeSmileCLI.execute</i> | Singolo progetto + callgraph=True | Analisi progetto e avvio generazione call graph |
| UT-CR02-49 | <i>CodeSmileCLI.execute</i> | Multiprogetto + callgraph=True | Call graph non generato; messaggio "non supportato" |
| UT-CR02-50 | <i>CodeSmileCLI.execute</i> | Multiprogetto sequenziale (multiple=True, parallel=False) | Analisi sequenziale + merge risultati + segnalazione completamento |
| UT-CR02-51 | <i>main()</i> | Argomenti validi (--input/--output) | Istanza CLI e avvia workflow (execute chiamato) |
| UT-CR02-52 | <i>main()</i> | Argomenti mancanti/non validi | Termina con SystemExit(1) |
| UT-CR02-53 | <i>if name == "main": main()</i> | Esecuzione modulo come script | Flusso "da script" eseguito correttamente |

3.1.2. CR03 – Call Graph (WebApp)

Sono stati aggiunti test di unità per componenti UI frontend (React) tramite Jest ed i test unitari lato WebApp coprono:

- rendering pagina;
- upload ZIP;
- rendering nodi (mock ReactFlow);
- apertura pop-up codice sorgente.

| | |
|---|------------------|
| Progetto: Code Smile | Versione: 1.0 |
| Documento: Post-Modification Testing Document | Data: 09/01/2026 |

3.2. Integration Test

Questa sezione descrive i test di integrazione introdotti per verificare il corretto funzionamento tra moduli o servizi distinti, con particolare attenzione al gateway WebApp ed all'integrazione della nuova funzionalità aggiuntiva relativa alla creazione del *call graph*.

3.2.1. CR02 – CLI -> call_graph.json

L'integrazione verifica l'intero flusso dalla CLI alla produzione del file *call_graph.json*, integrando CLI, *Analyzer* e *Generator*.

| ID | Descrizione | Input | Esito atteso |
|------------|--|---|---|
| IT-CR02-01 | Esecuzione completa CLI con generazione e Call Graph | CLI su progetto con 2 file e una dipendenza | File <i>call_graph.json</i> creato, nodi presenti e almeno un arco tra funzioni |

3.2.2. CR03 – Gateway → Static Analysis Service (Call Graph)

| ID | Descrizione | Input | Esito atteso |
|------------|--|---|--|
| IT-CR03-01 | Generazione Call Graph da snippet valido via Gateway | Richiesta POST con <i>code_snippet</i> e <i>include_call_graph=True</i> | HTTP 200, <i>success=True</i> , risposta contiene <i>call_graph</i> con nodi e archi |
| IT-CR03-02 | Gestione richiesta senza input | Richiesta POST senza file/snippet | HTTP 200, <i>success=False</i> (errore gestito) |

| | |
|---|------------------|
| Progetto: Code Smile | Versione: 1.0 |
| Documento: Post-Modification Testing Document | Data: 09/01/2026 |

3.1. System Test

I test di sistema preesistenti (20 scenari) sono stati rieseguiti con successo.

Per le modifiche effettuate è stata aggiunta una nuova categoria ai parametri di configurazione del tool del *category partition* specificato:

| CATEGORIA | SCELTE |
|--|--|
| Generazione del Call Graph (CG) | 1. True – generazione del Call Graph richiesta [se NP1, se CLI] 2. False – generazione del Call Graph non richiesta |

Nota: la generazione del call graph viene eseguita solo con l'analisi singolo progetto (NP1 = 1 progetto).

Di seguito sono elencati i nuovi test case legati alla funzionalità aggiunta. Specifichiamo solo i test case con CG1 = *True*, siccome quelli già implementati hanno implicitamente CG2 = *False*.

| Test Case ID | Test Frame | Oracolo atteso |
|--------------|--|---|
| TC_21 | NF2, EF1, NP1, SP1, NCS2, TCS1, ME2, EP2, NW0, RES1, OUT1, CG1, DB0, EG0 | Il tool completa l'analisi, genera il call graph e rileva un code smell di tipo generico. |
| TC_22 | NF2, EF1, NP1, SP1, NCS2, TCS2, ME2, EP2, NW0, RES1, OUT1, CG1, DB0, EG0 | Il tool completa l'analisi, genera il call graph e rileva un code smell di tipo API-Specific. |
| TC_23 | NF2, EF1, NP1, SP1, NCS3, TCS1, ME2, EP2, NW0, RES1, OUT1, CG1, DB0, EG0 | Il tool completa l'analisi, genera il call graph e rileva più code smell di tipo generico. |
| TC_24 | NF2, EF1, NP1, SP1, NCS3, TCS2, ME2, EP2, NW0, RES1, OUT1, CG1, DB0, EG0 | Il tool completa l'analisi, genera il call graph e rileva più code smell di tipo API-Specific. |
| TC_25 | NF2, EF1, NP1, SP1, NCS3, TCS3, ME2, EP2, NW0, RES1, OUT1, CG1, DB0, EG0 | Il tool completa l'analisi, genera il call graph e rileva più code smell di tipo misto. |
| TC_26 | NF2, EF1, NP1, SP1, NCS2, TCS1, ME2, EP1, NW2, OUT1, CG1, DB0, EG0 | Il tool completa l'analisi, rileva un code smell di tipo generico, tramite analisi parallela, con numero di walkers < 5 e genera correttamente il call graph. |
| TC_27 | NF2, EF1, NP1, SP1, NCS2, TCS1, ME2, EP1, NW3, OUT1, CG1, DB0, EG0 | Il tool completa l'analisi, rileva un code smell di tipo generico, tramite analisi parallela, con numero di walkers = 5 e genera correttamente il call graph. |
| TC_28 | NF2, EF1, NP1, SP1, NCS2, TCS1, ME2, EP1, NW4, OUT1, CG1, DB0, EG0 | Il tool completa l'analisi, rileva un code smell di tipo generico, tramite analisi parallela, con numero di walkers > 5 e genera correttamente il call graph. |

| | |
|---|------------------|
| Progetto: Code Smile | Versione: 1.0 |
| Documento: Post-Modification Testing Document | Data: 09/01/2026 |

3.1. End-to-End Testing (WebApp)

Per la CR03 è stata condotta una suite di test End-to-End (E2E) dedicata alla funzionalità di generazione e visualizzazione del *call graph*. Utilizzando *Cypress*, è stato verificato il comportamento dell'interfaccia *React* durante l'intero ciclo di vita dell'analisi, dal caricamento del progetto alla visualizzazione interattiva dei nodi.

I test *E2E* includono:

- **Rendering dell'interfaccia di upload:** Verifica che, accedendo alla pagina */call-graph*, l'utente visualizzi correttamente l'area di caricamento ("Drag & Drop" o click) pronta per ricevere file ZIP, file Python oppure un folder.
- **Upload e Visualizzazione del Grafo:** Simulazione del caricamento di un archivio *.zip* e verifica della corretta invocazione dell'API. Il test controlla che, al termine dell'analisi, il componente *React Flow* renderizzi correttamente i nodi e gli archi di dipendenza nello spazio di lavoro.
- **Interazione con i Nodi:** Verifica dell'interattività del grafo. Cliccando su uno specifico nodo il test assicura l'apertura di una finestra modale.

4. Conclusione

Tutte le attività di **Post-Modification Testing** e **Regression Testing** sono state completate con successo. Le nuove funzionalità introdotte tramite le Change Requests risultano correttamente integrate e operative, mentre il comportamento del sistema preesistente è stato preservato. Non sono state riscontrate regressioni né malfunzionamenti.

In particolare, il Post-Modification Testing ha validato le funzionalità introdotte ed estese da:

- **CR01:** allineamento dinamico di endpoint e import tra esecuzione locale e ambiente containerizzato;
- **CR02:** generazione del *Call Graph* nella CLI, inclusa l'introduzione dei nuovi System Test TC_21–TC_28;
- **CR03:** generazione e visualizzazione del *Call Graph* nella WebApp, con verifiche sia lato backend sia lato frontend.

Per quanto riguarda la regressione, la campagna di test eseguita riporta esito SUPERATO, con 287/287 test passati (con le esclusioni previste per i moduli AI e GUI), confermando la stabilità complessiva del sistema dopo le modifiche.