UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica

Corso di Laurea Magistrale in Informatica

IMPACT ANALYSIS

Ingegneria del Software: Tecniche Avanzate - Impact Analysis

Luigi Auriemma

Matricola: NF22500161

Ivan Chiello

Matricola: NF22500167

Repository: `https://github.com/LuigiAuriemma/smell_ai.git`

Anno Accademico 2025-2026


sesa
lab
SOFTWARE ENGINEERING
SALERNO

Indice

Elenco delle figure

Elenco delle tabelle

CAPITOLO 1

Introduzione

CodeSmile è un software ideato per l'analisi automatizzata della qualità del codice in progetti di Machine Learning, con un focus specifico sull'identificazione di pattern inefficienti (Code Smells). L'architettura del sistema combina tecniche di analisi statica (AST parsing) e modelli basati su LLM per offrire agli sviluppatori strumenti di refactoring mirati.
Questo documento di Impact Analysis si propone di esaminare in via preventiva le modifiche richieste (Change Requests) per l'evoluzione della piattaforma. Lo scopo dell'analisi è stimare l'estensione degli interventi necessari sui moduli esistenti, permettendo così di pianificare lo sviluppo in modo accurato e di mitigare i rischi legati a potenziali effetti collaterali o regressioni nel codice. L'indagine viene svolta attraverso un'ispezione statica dei sorgenti, volta a tracciare le dipendenze logiche tra i componenti.

1.1 Obiettivi del documento

L'analisi persegue i seguenti obiettivi operativi:

- **Mappatura Architetturale:** Ricostruire la struttura delle dipendenze del sistema CodeSmile attraverso l'analisi statica, identificando i moduli critici e le

interconnessioni tra servizi.

- **Stima dell'Impatto:** Determinare per ogni Change Request l'insieme dei file che necessitano di modifica diretta e quelli esposti a rischio di propagazione.
- **Giustificazione Formale:** Fornire evidenze che motivino l'inclusione dei vari componenti negli insiemi di impatto previsti.
- **Verifica a Posteriori:** Definire i criteri quantitativi (Metriche di Precision e Recall) per valutare, a fine implementazione, la discrepanza tra l'impatto ipotizzato in questa sede e le modifiche effettivamente realizzate.

1.2 Change Request Analizzate

Le richieste di modifica analizzate sono:

ID	Titolo	Descrizione Sintetica	Priorità
CR01	Creazione di un Call graph per l'analisi delle dipendenze nella CLI	Potenziamento del motore di analisi statica (CLI) per generare grafi di chiamata (Call Graph) dei file analizzati, permettendo di visualizzare la propagazione strutturale degli smell rilevati.	Alta
CR02	Visualizzazione del Call Graph con Code Smells nella Web App	Estensione della Web Application con un'interfaccia grafica interattiva dedicata alla navigazione dei Call Graph prodotti dall'analisi.	Media

Tabella 1.1: Tabella riassuntiva delle Change Request

1.3 Terminologia e Acronimi

Per classificare i componenti software in base al loro coinvolgimento nelle modifiche, si adottano le seguenti definizioni standard:

- **SIS (Starting Impact Set):** Identifica i componenti che saranno oggetto di modifica diretta per soddisfare i requisiti della Change Request.
- **CIS (Candidate Impact Set):** Comprende l'insieme esteso dei componenti che, pur non essendo modificati direttamente, potrebbero subire effetti indesiderati a causa di relazioni di dipendenza (es. chiamate a funzione, import, ereditarietà) con gli elementi del SIS.
- **AIS (Actual Impact Set):** Rappresenta l'insieme dei componenti effettivamente alterati al termine della fase di implementazione.
- **FPIS (False Positive Impact Set):** Sottoinsieme del CIS/SIS che, contrariamente alle previsioni, non ha richiesto interventi.
- **DIS (Discovered Impact Set):** Componenti che hanno richiesto modifiche impreviste durante lo sviluppo, sfuggiti all'analisi iniziale.

CAPITOLO 2

Change Request 01

2.1 Descrizione della Change Request

Titolo	Creazione di un Call graph per l'analisi delle dipendenze dei file contenenti smell nella CLI
Situazione Attuale	L'attuale strumento da riga di comando (CLI) permette di avviare l'analisi statica per l'individuazione di code smell, producendo un report tabellare (CSV). L'analisi è limitata alla singola unità di codice (file o funzione) e manca una rappresentazione strutturale delle dipendenze tra i moduli analizzati.
Situazione Desiderata	Estensione della CLI con una nuova opzione (flag) che abiliti la generazione automatica di un *Call Graph*. Il sistema dovrà costruire e restituire una rappresentazione delle chiamate tra le funzioni analizzate, permettendo di visualizzare come gli smell rilevati si distribuiscono nella topologia del progetto.

Tabella 2.1: Dettaglio Change Request CR01

2.2 Analisi Statica e Architetturale

L'analisi statica del sistema attuale evidenzia una struttura gerarchica lineare per l'esecuzione dell'analisi. Il punto di ingresso è gestito dal modulo `cli_runner.py`, che istanzia e configura il `ProjectAnalyzer`. Quest'ultimo orchestra l'iterazione sui file del progetto e delega l'ispezione del codice sorgente (parsing AST e verifica regole) alla classe `Inspector`.

2.2.1 Call Graph Architetturale (Stato Attuale)

Di seguito è riportata la struttura delle chiamate tra le classi principali coinvolte nel flusso di esecuzione della CLI, ricostruita tramite analisi del codice sorgente fornito.

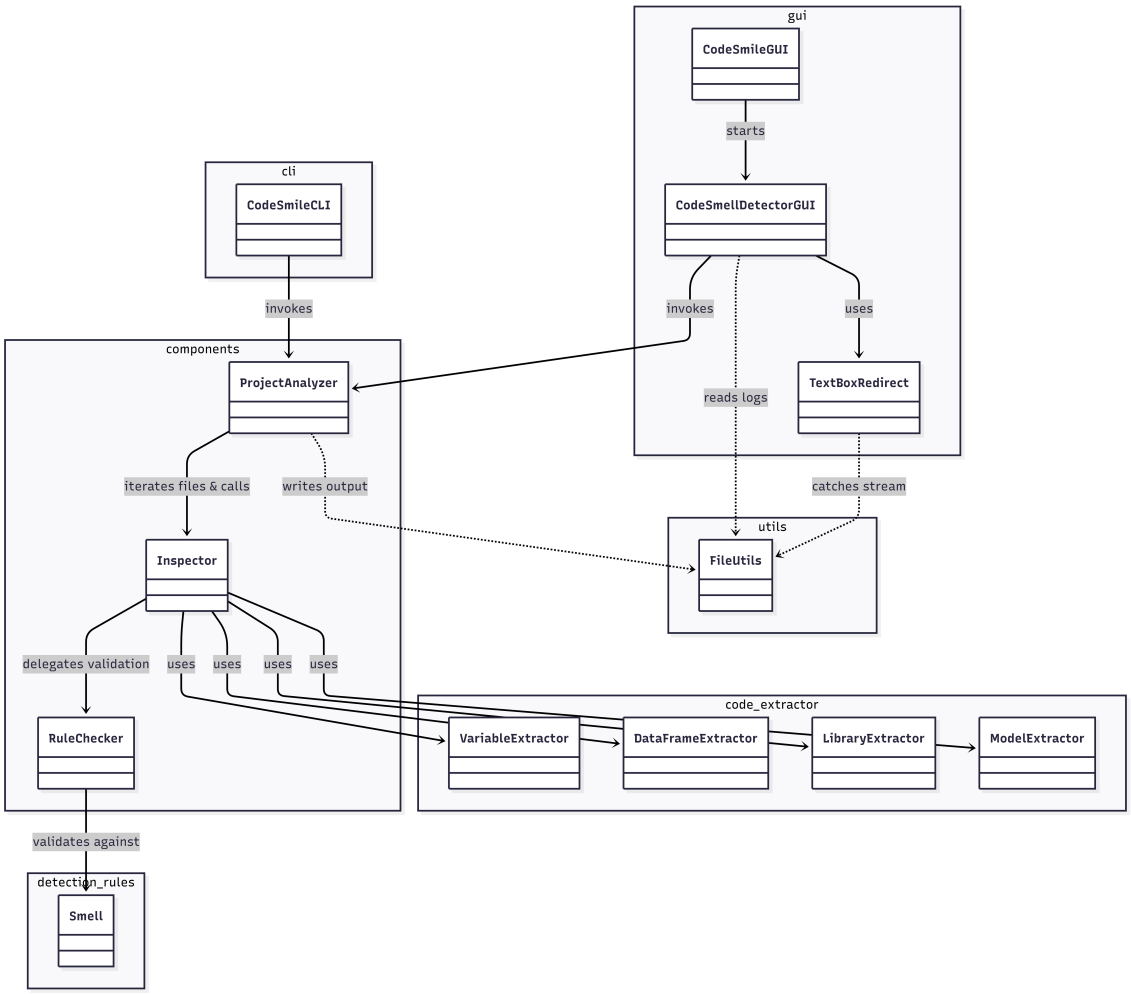

Figura 2.1: Architettura Attuale (As-Is)

2.2.2 Matrice di Raggiungibilità

La matrici seguenti rappresentano la raggiungibilità tra le classi principali dell'attuale sistema. I colori rappresentano la distanza topologica (numero di chiamate necessarie per raggiungere un componente).

Sorgente \ Destinazione	CS.DetectorGUI	CodeSmileCLI	CodeSmileGUI	Df.Extractor	FileUtils	Inspector
CodeSmellDetectorGUI	0	-	-	3	2	2
CodeSmileCLI	-	0	-	3	2	2
CodeSmileGUI	1	-	0	4	3	3
DataFrameExtractor	-	-	-	0	-	-
FileUtils	-	-	-	-	0	-
Inspector	-	-	-	1	-	0
ProjectAnalyzer	-	-	-	2	1	1
VariableExtractor	-	-	-	-	-	-

Sorgente \ Destinazione	Lib.Extractor	Mod.Extractor	Proj.Analyzer	RuleChecker	Smell	Var.Extractor
CodeSmellDetectorGUI	3	3	1	3	4	3
CodeSmileCLI	3	3	1	3	4	3
CodeSmileGUI	4	4	2	4	5	4
DataFrameExtractor	-	-	-	-	-	-
FileUtils	-	-	-	-	-	-
Inspector	1	1	-	1	2	1
ProjectAnalyzer	2	2	0	2	3	2
VariableExtractor	-	-	-	-	-	0

Tabella 2.2: Matrice di Raggiungibilità tra le componenti del sistema (Gradiente Viola)

2.3 Impact Analysis

2.3.1 Start Impact Set (SIS)

Lo Start Impact Set (SIS) identifica le entità software che richiedono una modifica diretta per implementare i requisiti descritti nella Change Request. Per la CR01, sono stati individuati tre file critici che costituiscono il punto di iniezione della modifica.

File (Percorso)	Ruolo	Motivazione dell'Inclusione
components/inspector.py	Logica Core	**Estensione del Parsing AST**: Attualmente il file analizza le entità in isolamento. Deve essere modificato per includere nuovi metodi *visitor* (`visit_Call`, `visit_Import`) necessari per intercettare le dipendenze verso moduli esterni.
components/project_analyzer.py	Controller	**Aggregazione Dati**: Gestisce il ciclo di vita dell'analisi. La modifica è necessaria per trasformare l'iterazione sequenziale sui file in un processo a due fasi: analisi locale e successiva costruzione del grafo globale (linking).
cli/cli_runner.py	Entry Point	**Interfaccia Utente**: Il file deve essere aggiornato per gestire nuovi argomenti da riga di comando (es. flag `-graph`) e propagare la configurazione ai componenti sottostanti.

Tabella 2.3: Start Impact Set (SIS) per la generazione del Call Graph

$$SIS = \left\{ \begin{array}{l} \texttt{cli/cli\_runner.py,components/inspector.py,} \\ \texttt{components/project\_analyzer.py} \end{array} \right\}$$

2.3.2 Candidate Impact Set (CIS)

L'analisi del Candidate Impact Set (CIS) mira a identificare i componenti che dipendono, direttamente o indirettamente, dagli elementi del SIS. Inizialmente, l'indagine è stata condotta tramite ricerca testuale delle invocazioni e delle importazioni nel codice sorgente utilizzando il comando `Select-String`.
Tuttavia, data la natura puramente sintattica e la relativa bassa affidabilità di tale metodo (che può generare falsi positivi), si è proceduto a una revisione sommaria manuale delle dipendenze logiche e degli accoppiamenti tra i moduli.
A seguito di questa analisi , è stato determinato che il CIS coincide interamente con il SIS ($CIS = SIS$).

2.4 Implementazione

2.4.1 Actual Impact Set (AIS)

Durante la fase di implementazione, le decisioni tecniche volte a preservare la modularità del sistema hanno portato a una leggera divergenza tra l'Impact Set stimato (SIS) e quello effettivo (AIS).
Mentre il SIS prevedeva una modifica invasiva del file `components/inspector.py` per includere la logica di estrazione astrazione, l'approccio adottato ha privilegiato l'estensione tramite nuovi moduli dedicati. Di conseguenza, l'AIS comprende i seguenti elementi:

- **File Modificati (dal SIS)**:
  - `cli/cli_runner.py`: Aggiornato per gestire il flag CLI.
  - `components/project_analyzer.py`: Aggiornato per orchestrare il builder del grafo.
- **Nuovi File Creati (Estensione)**:
  - `components/dependency_graph_builder.py`: Contiene la logica di costruzione e salvataggio del grafo, sostituendo le modifiche previste in `inspector.py`.

- `code_extractor/call_graph_extractor.py`: Implementa il visitatore AST specifico per le chiamate di funzione.

Formalmente, l'insieme dei file impattati risulta essere:

$$AIS_{CR01} = \left\{ \begin{array}{l} \texttt{cli/cli\_runner.py}, \\ \texttt{components/project\_analyzer.py}, \\ \texttt{components/dependency\_graph\_builder.py (New)}, \\ \texttt{code\_extractor/call\_graph\_extractor.py (New)} \end{array} \right\}$$

Questa discrepanza tra SIS e AIS evidenzia un approccio *additivo* piuttosto che *modificativo* per la logica core, riducendo il rischio di regressioni sui componenti esistenti come l'`inspector.py`.

2.4.2 False Positive Impact Set (FPIS)

Il False Positive Impact Set (FPIS) è costituito dalle classi che erano state inizialmente stimate come impattate dalla modifica (membri del SIS), ma che non sono state effettivamente modificate durante l'implementazione (non membri dell'AIS). La tabella seguente descrive queste classi e la giustificazione per la loro esclusione dalla modifica.

Classe/File	**Motivo dell'Esclusione**
`components/inspector.py`	Sebbene inizialmente identificata come la posizione per l'avvio della generazione del call graph, abbiamo optato per implementare la logica in una nuova classe dedicata (`DependencyGraphBuilder`) invece di modificare l'`Inspector` esistente. Questo approccio aderisce al "Single Responsibility Principle", mantenendo il focus dell'`Inspector` esclusivamente sulle regole di rilevamento dei code smell.

Tabella 2.4: False Positive Impact Set (FPIS) per la CR-01

Analisi: L'esclusione di `inspector.py` dalle modifiche finali implica un'architettura più pulita rispetto a quanto originariamente stimato. Delegando la complessa

logica di attraversamento dell'AST al nuovo modulo `code_extractor` e l'orchestrazione al `DependencyGraphBuilder`, abbiamo evitato di introdurre rischi di regressione nella logica core di rilevamento degli smell.

2.4.3 Discovered Impact Set (DIS)

Il Discovered Impact Set (DIS) comprende le classi o i file che non erano stati inclusi nell'insieme di impatto iniziale (SIS), ma che sono stati modificati o creati durante l'effettiva implementazione della Change Request. Questi componenti rappresentano il lavoro "scoperto" durante la fase di sviluppo, spesso dovuto a decisioni architetturali prese in corso d'opera per migliorare la manutenibilità del codice.
Le seguenti classi compongono il DIS per la CR-01:

Classe/File	**Motivazione dell'Inclusione**
`components/ dependency_graph_builder.py`	È stata creata questa nuova classe per orchestrare la costruzione del grafo delle dipendenze. Invece di sovraccaricare `inspector.py` (che è finito nel FPIS), abbiamo delegato a questo componente la responsabilità di collegare l'estrattore alle logiche di analisi del progetto.
`code_extractor/ call_graph_extractor.py`	È stato necessario implementare un visitatore AST specifico (basato su `ast.NodeVisitor`) per identificare le chiamate a funzione e le definizioni all'interno del codice sorgente, separando questa logica dall'analisi generica dei progetti.

Tabella 2.5: Discovered Impact Set (DIS) per la CR-01

Analisi: La presenza di questi nuovi file nel DIS, bilanciata dalla rimozione di `inspector.py` (FPIS), dimostra che l'impatto della modifica si è spostato dall'*alterazione* di codice esistente alla *creazione* di nuovi moduli modulari. Questo riduce il rischio di regressioni, isolando la nuova funzionalità in file dedicati.

2.5 Metrics

In questa sezione valutiamo l'accuratezza della stima dell'impatto utilizzando le metriche di *Precision* e *Recall*.

- **Candidate Impact Set (CIS/SIS):** 3 classi
- **Actual Impact Set (AIS):** 4 classi
- **Intersezione (CIS ∩ AIS):** 2 classi

Precision (Precisione)

Misura la frazione di classi previste che sono state effettivamente modificate.

$$Precision = \frac{|CIS \cap AIS|}{|CIS|} = \frac{2}{3} \approx 0,67 \quad (67\%)$$

Un valore del 67% indica che la maggior parte delle classi previste è stata effettivamente coinvolta, con un solo "falso positivo" (`inspector.py`).

Recall (Richiamo)

Misura la frazione di classi effettivamente modificate che erano state previste correttamente all'inizio.

$$Recall = \frac{|CIS \cap AIS|}{|AIS|} = \frac{2}{4} = 0,50 \quad (50\%)$$

Un valore del 50% indica che metà del lavoro effettivo (l'implementazione delle nuove classi `DependencyGraphBuilder` e `CallGraphExtractor`) non era stata prevista nella stima iniziale, evidenziando la scoperta di nuove necessità architetturali durante lo sviluppo (DIS).

CAPITOLO 3

Change Request 02

3.1 Descrizione della Change Request

Titolo	Visualizzazione del Call Graph con Code Smells nella Web App
Situazione Attuale	La Web Application esistente presenta i risultati dell'analisi esclusivamente tramite liste statiche. La dashboard attuale (Frontend React) recupera i dati dall'API ma non offre strumenti per esplorare visivamente le relazioni strutturali tra i file analizzati, rendendo difficile comprendere la propagazione degli smell.
Situazione Desiderata	Integrazione nella dashboard web di un visualizzatore di grafi interattivo. L'utente dovrà visualizzare i nodi (file) e gli archi (chiamate), con evidenziazione grafica (color-coding) per i componenti contenenti smell. Cliccando su un nodo, il sistema dovrà mostrare i dettagli dell'analisi specifica nel pannello laterale.

Tabella 3.1: Dettaglio Change Request CR02

3.2 Analisi Statica e Architetturale (Web Layer)

L'architettura della funzionalità Web differisce sostanzialmente da quella della CLI per la natura distribuita e asincrona dei componenti. Il sistema attuale implementa un pattern a microservizi orchestrato tramite Docker, dove la responsabilità dell'analisi è segregata dal layer di presentazione.
L'analisi statica del codice sorgente Web (`webapp/`) ha permesso di ricostruire la catena di dipendenze che trasforma una richiesta utente in un'esecuzione del backend.

3.2.1 Call Graph Architetturale (Stato Attuale)

Per comprendere l'impatto della visualizzazione del grafo, è stata mappata la pipeline di esecuzione "End-to-End". A differenza di una vista gerarchica, il diagramma seguente adotta una prospettiva orientata al **Data Flow**, evidenziando il disaccoppiamento tra i servizi.

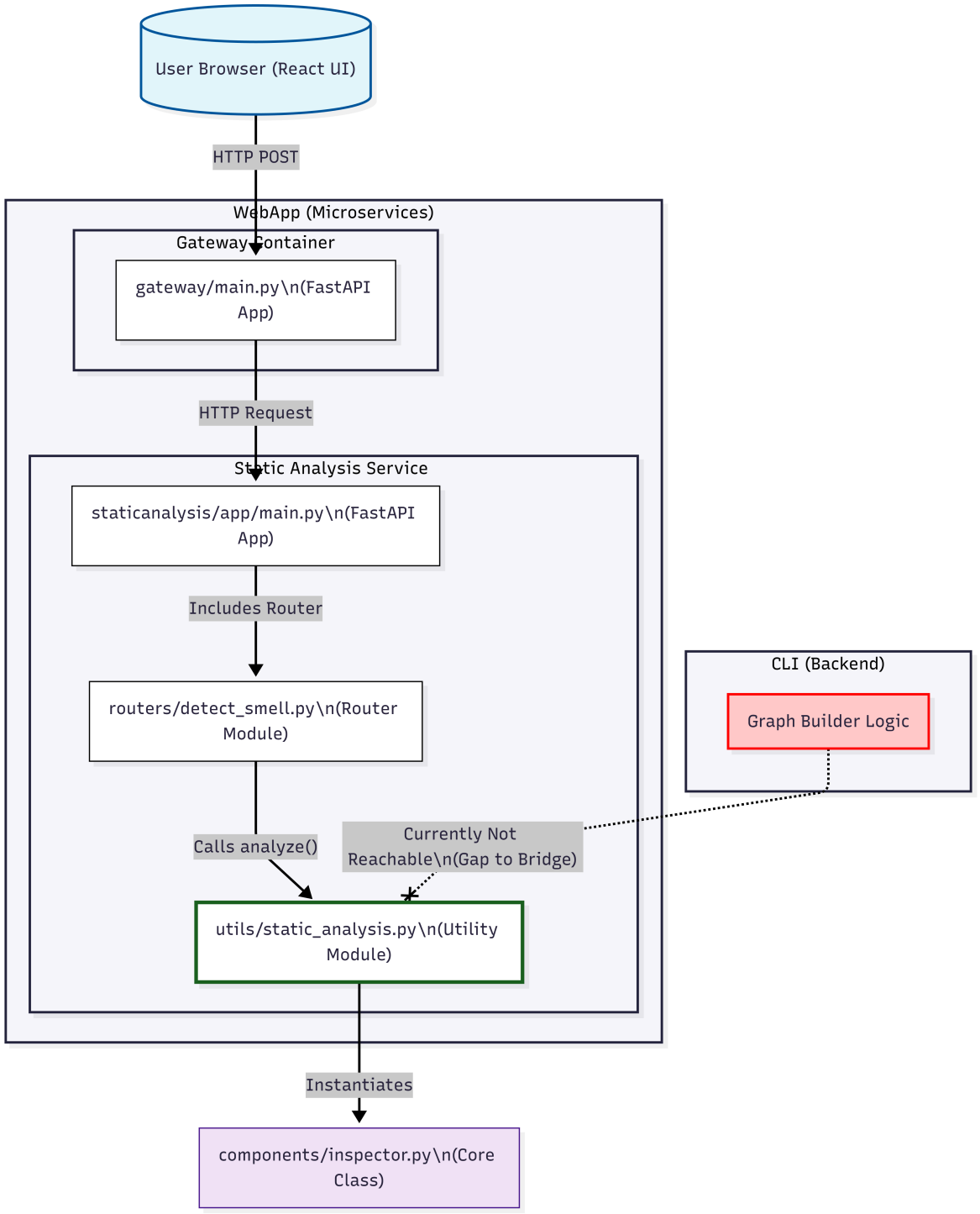

Figura 3.1: Pipeline di Servizio As-Is: Dal Frontend al Core Backend

3.2.2 Matrice di Raggiungibilità (Web Services Layer)

La seguente matrice quantifica la distanza logica tra i layer dell'architettura web. Nel contesto della CR02, una distanza elevata dal Frontend indica una maggiore

complessità di propagazione dei dati: il nuovo JSON del grafo deve risalire l'intera catena (Inspector → UI) attraversando diverse barriere di serializzazione.

Sorgente \ Destinazione	Frontend UI	API Gateway	Smell Router	Static Adapter	Inspector (Core)
Frontend UI	0	1	2	3	4
API Gateway	1	0	1	2	3
Smell Router	2	1	0	1	2
Static Adapter	-	-	1	0	1
Inspector (Core)	-	-	-	-	0

Tabella 3.2: Matrice di Raggiungibilità Cross-Layer (Web App)

3.3 Impact Analysis

3.3.1 Start Impact Set (SIS)

Lo Start Impact Set per la CR02 è stato esteso per includere tutti i componenti che partecipano alla definizione dei contratti di interfaccia (Schemas) e al trasporto dati. L'identificazione accurata dei file TypeScript (Frontend) e Pydantic (Backend) è necessaria per garantire la coerenza dei tipi di dato lungo la catena.

File (Percorso)	Tipo	Motivazione Specifica
`webapp/app/call-graph/pages.tsx` `webapp/app/page.tsx`	Frontend	**UI/UX**: Creazione della nuova view di visualizzazione (Next.js Page) e aggiunta del punto di navigazione (link) nella Home Page esistente.
`webapp/utils/api.ts`	Frontend	**Client Http**: Aggiornamento delle definizioni di tipo TypeScript (interfaces) per gestire il payload JSON esteso contenente i nodi e gli archi del grafo.
`webapp/gateway/main.py`	Proxy	**Data Forwarding**: Il Gateway deve essere verificato per assicurare che non filtri o tronchi il nuovo payload complesso proveniente dal microservizio di analisi.
`webapp/services/staticanalysis/app/routers/detect_smell.py`	Endpoint	**Controller**: Orchestrazione della chiamata. Deve invocare l'utility aggiornata e mappare il risultato nel nuovo schema di risposta.
`webapp/services/staticanalysis/app/utils/static_analysis.py`	Logic	**Core Integration**: Punto di contatto con il backend Python (Inspector). Qui avviene la generazione fisica della struttura dati del grafo.
`webapp/services/staticanalysis/app/schemas/*.py` (req/res)	Model	**Validazione**: Modifica delle classi Pydantic per includere formalmente il campo `call_graph` nei contratti API (Input/Output), prevenendo errori di validazione "422 Unprocessable Entity".

$$SIS = \left\{ \begin{array}{l} \texttt{frontend/pages/*,frontend/utils/api.ts,} \\ \texttt{gateway/main.py,staticanalysis/routers/*,} \\ \texttt{staticanalysis/utils/*,staticanalysis/schemas/*} \end{array} \right\}$$

3.3.2 Candidate Impact Set (CIS)

Il Candidate Impact Set (CIS) identifica i componenti che potrebbero subire effetti collaterali a causa delle modifiche introdotte nel SIS. L'analisi iniziale, condotta tramite strumenti di ricerca testuale (`Select-String`), aveva suggerito potenziali propagazioni verso i file di configurazione globale
Tuttavia, a seguito di una revisione manuale, resa necessaria dalla bassa affidabilità intrinseca dei metodi di ricerca puramente sintattici, è stato confermato che tali dipendenze non costituiscono un rischio reale. Di conseguenza, si determina che:

$$CIS = SIS$$

Questa conclusione è supportata dalle seguenti evidenze tecniche:

- **Isolamento Funzionale**: La logica di generazione del grafo è confinata interamente nel dominio dell'analisi statica. Non si ravvisano effetti collaterali su altri microservizi (es. AI Service o Report Service), né la necessità di alterare il file `main.py` per registrare nuovi middleware o dipendenze globali.

L'uguaglianza tra CIS e SIS semplifica notevolmente il piano di intervento, riducendo il rischio di regressione e confermando l'efficacia dell'incapsulamento del design attuale.

3.4 Implementazione

3.4.1 Actual Impact Set (AIS)

Durante l'implementazione della CR02 nel contesto della webapp a microservizi, le scelte architetturali hanno privilegiato la separazione delle responsabilità, portando a una divergenza tra l'Impact Set stimato (SIS) e quello effettivo (AIS).

Mentre il SIS ipotizzava l'integrazione della nuova logica all'interno dei router e degli schemi esistenti (es. `detect_smell.py`), l'approccio adottato ha puntato sulla creazione di componenti verticali dedicati, dal frontend al backend. Di conseguenza, l'AIS comprende i seguenti elementi:

- **File Modificati (dal SIS)**:
  - `webapp/app/page.tsx`: Aggiornato per includere il punto di navigazione nella Dashboard.
  - `webapp/gateway/main.py`: Aggiornato per aggiungere la rotta di inoltro al microservizio di analisi statica.
  - `webapp/services/staticanalysis/app/main.py`: Aggiornato per registrare il nuovo router nell'applicazione FastAPI.
- **Nuovi File Creati (Estensione)**:
  - `webapp/app/call-graph/page.tsx`: Pagina frontend dedicata alla visualizzazione.
  - `webapp/components/CallGraphVisualizer.tsx`: Componente React che implementa la logica di rendering del grafo.
  - `webapp/services/staticanalysis/app/routers/call_graph.py`: Gestisce l'endpoint di generazione, sostituendo la modifica prevista a `detect_smell.py`.
  - `webapp/services/staticanalysis/app/schemas/graph_schemas.py`: Definisce i contratti dati specifici per nodi e archi.

Formalmente, l'insieme dei file impattati risulta essere:

$$AIS_{CR02} = \begin{cases} \texttt{webapp/app/page.tsx}, \\ \texttt{webapp/gateway/main.py}, \\ \texttt{webapp/services/staticanalysis/app/main.py}, \\ \texttt{webapp/app/call-graph/page.tsx (New)}, \\ \texttt{webapp/components/CallGraphVisualizer.tsx (New)}, \\ \texttt{webapp/services/staticanalysis/routers/call\_graph.py (New)}, \\ \texttt{webapp/services/staticanalysis/schemas/graph\_schemas.py (Ne} \end{cases}$$

Questa discrepanza conferma l'adozione di un pattern "Extension" piuttosto che "Modification", garantendo che la nuova funzionalità di visualizzazione non introduca rischi di regressione sugli endpoint di rilevamento degli smell già esistenti.

3.4.2 False Positive Impact Set (FPIS)

Il False Positive Impact Set (FPIS) evidenzia i componenti che erano stati inclusi nell'Impact Set iniziale (SIS) come candidati alla modifica, ma che sono rimasti inalterati durante l'effettiva implementazione. Queste esclusioni sono il risultato di decisioni progettuali volte a mantenere pulita l'architettura a microservizi.

Classe/File	Motivo dell'Esclusione
`webapp/utils/api.ts`	Inizialmente si prevedeva di estendere le funzioni API esistenti in questo file condiviso. Tuttavia, dato che la logica di chiamata per il grafo richiedeva una gestione degli errori e un timeout specifici, si è preferito incapsulare la chiamata `fetch` direttamente nel nuovo componente `CallGraphVisualizer.tsx`, lasciando `api.ts` invariato per le funzioni core.
`webapp/services/staticanalysis/app/routers/detect_smell.py`	Si ipotizzava di aggiungere qui l'endpoint per il call graph. Per rispettare il principio di separazione delle responsabilità (SRP) e non appesantire il controller principale di rilevamento, l'endpoint è stato invece spostato nel nuovo router dedicato `call_graph.py`.
`webapp/services/staticanalysis/app/utils/static_analysis.py`	La logica core di generazione del grafo non ha richiesto modifiche a questo file di utilità esistente, in quanto la libreria sottostante (`dependency_graph_builder`) è stata importata e utilizzata direttamente dal nuovo router, senza dover adattare le utility esistenti.
`schemas/requests.py`	Invece di modificare gli schemi di richiesta esistenti rendendoli ibridi, si è scelto di riutilizzare lo schema base solo in lettura e definire nuovi schemi di risposta in un file separato (`graph_schemas.py`) per mantenere puliti i contratti d'interfaccia.

Tabella 3.4: False Positive Impact Set (FPIS) per la CR-02

Analisi: L'elevato numero di elementi nel FPIS indica una sovrastima iniziale della necessità di accoppiamento. L'architettura del sistema ha permesso di aggiungere la

funzionalità in modo ortogonale, riducendo drasticamente la necessità di modificare codice legacy.

3.4.3 Discovered Impact Set (DIS)

Il Discovered Impact Set (DIS) comprende le classi o i file che non erano inclusi nell'insieme di impatto iniziale (SIS), ma che sono stati creati o modificati durante l'effettiva implementazione della richiesta di modifica. Questi elementi rappresentano componenti nuovi introdotti per garantire una migliore modularità e manutenibilità del sistema distribuito.
Le seguenti componenti formano il DIS per la CR-02:

Classe/File	Motivazione dell'Inclusione
`webapp/app/call-graph/page.tsx`	È stata creata una rotta dedicata nel framework Next.js. Invece di modificare la dashboard esistente per ospitare il grafo (come inizialmente ipotizzato), si è scelto di creare una pagina indipendente per gestire meglio lo stato dell'applicazione e il rendering lato client.
`webapp/components/CallGraphVisualizer.tsx`	È emersa la necessità di un componente React specializzato per incapsulare la logica complessa di visualizzazione (con `react-plotly.js`) e le interazioni utente, separandole dalla logica di routing della pagina.
`webapp/services/staticanalysis/app/routers/call_graph.py`	Per evitare di violare il Single Responsibility Principle nel router di rilevamento smell esistente, è stato creato un nuovo controller dedicato esclusivamente alla gestione delle richieste di generazione del grafo e all'orchestrazione del backend.
`webapp/services/staticanalysis/app/schemas/graph_schemas.py`	Durante lo sviluppo è emerso che i modelli dati esistenti non erano sufficienti per descrivere la struttura nidificata nodi-archi del grafo JSON. È stato quindi necessario definire nuovi schemi Pydantic specifici.

Tabella 3.5: Discovered Impact Set (DIS) per la CR-02

Analisi: La presenza significativa di nuovi file nel DIS, speculare agli elementi del FPIS, conferma che l'implementazione ha seguito una strategia di "espansione" del codice base piuttosto che di modifica, garantendo un isolamento ottimale della nuova feature.

3.4.4 Metrics

Valutiamo ora l'accuratezza della stima dell'impatto per questa Change Request utilizzando le metriche standard.

- **Start Impact Set (SIS):** 6 file
- **Actual Impact Set (AIS):** 7 file
- **Intersezione (CIS ∩ AIS):** 2 file

Precision (Precisione)

Indica quanto la stima iniziale fosse "mirata", ovvero quanta parte del lavoro previsto è stata effettivamente realizzata su quei file specifici.

$$Precision = \frac{|CIS \cap AIS|}{|CIS|} = \frac{2}{6} \approx 0,33 \quad (33\%)$$

Un valore del 33% indica una precisione bassa. Questo è dovuto alla scelta architetturale consapevole di non modificare i componenti esistenti (che costituivano la maggior parte del SIS) ma di crearne di nuovi (che sono finiti nel DIS). La stima iniziale prevedeva un approccio molto più invasivo di quanto poi realizzato.

Recall (Richiamo)

Indica quanto del lavoro finale effettivamente svolto era stato previsto correttamente all'inizio.

$$Recall = \frac{|CIS \cap AIS|}{|AIS|} = \frac{2}{7} \approx 0,29 \quad (29\%)$$

Un valore del 29% conferma che la gran parte dell'implementazione (circa il 70%) ha riguardato file nuovi o componenti imprevisti (i membri del DIS come il Visualizer o il Router dedicato). Questo riflette un cambio di strategia in corso d'opera verso una maggiore modularità.