Università degli Studi di Salerno
Corso di Ingegneria del Software: Tecniche Avanzate

Code Smile
Impact Analysis Document
Versione 1.0

Team:
Antonio Caiazzo
Emanuele Iovane
Salvatore Di Martino

Partecipanti:

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

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

Indice

1. Introduzione 4
1.1. Obiettivo 4
1.2. Descrizione delle Change Request 5
1.3. Definizioni e acronimi 5
2. Change Request CR01 6
2.1. Analisi statica 6
2.2. Impact Analysis 7
2.2.1. SIS (Start Impact Set) 7
2.2.2. CIS (Candidate Impact Set) 7
2.3. Implementazione della Change Request 8
2.3.1. AIS (Actual Impact Set) 8
2.3.2. FPIS (False Positive Impact Set) 8
2.3.3. DIS (Discovered Impact Set) 8
2.4. Metriche 8
3. Change Request CR02 9
3.1. Analisi statica 9
3.2. Impact Analysis 10
3.2.1. SIS (Start Impact Set) 10
3.2.2. CIS (Candidate Impact Set) 10
3.3. Implementazione della Change Request 11
3.3.1. AIS (Actual Impact Set) 11
3.3.2. FPIS (False Positive Impact Set) 11
3.3.3. DIS (Discovered Impact Set) 11
3.4. Metriche 11
4. Change Request CR03 12
4.1. Analisi statica 12
4.2. Impact Analysis 13
4.2.1. SIS (Start Impact Set) 13
4.2.2. CIS (Candidate Impact Set) 13
4.3. Implementazione della Change Request 14
4.3.1. AIS (Actual Impact Set) 14
4.3.2. FPIS (False Positive Impact Set) 14
4.3.3. DIS (Discovered Impact Set) 14
4.4. Metriche 14

1. Introduzione

Il presente documento di *Impact Analysis* descrive e valuta l'impatto delle Change Request (CR) previste per il progetto Code Smile, con l'obiettivo di supportare decisioni di evoluzione e refactoring riducendo il rischio di effetti collaterali. L'analisi è condotta tramite analisi statica della codebase, finalizzata a ricostruire la struttura del sistema a livello di moduli, classi e servizi, e a identificare relazioni di dipendenza che influenzano la propagazione delle modifiche.

Per rappresentare in modo chiaro e verificabile tali dipendenze, è stato adottato il call graph come artefatto principale di supporto: esso permette di (i) evidenziare il flusso di invocazioni e l'accoppiamento tra componenti, (ii) individuare nodi centrali e punti di orchestrazione (es. gateway, moduli di analisi, generatori), (iii) derivare in modo sistematico insiemi di impatto (SIS/CIS) e ridurre omissioni, e (iv) motivare le scelte implementative confrontando architettura prevista e architettura effettiva. In questo documento, quando necessario per la leggibilità, il call graph viene presentato a un livello di astrazione coerente con la CR (architetturale tra servizi o tra classi principali), evitando dettagli a grana troppo fine che renderebbero meno interpretabile l'analisi.

A completamento del call graph, viene utilizzata una matrice di raggiungibilità tra i componenti principali (o classi principali, a seconda della CR) già mostrati nel call graph. Ogni cella *i,j* assume i seguenti valori:

- **0**: assenza di dipendenza tra la classe/componente *i* e la classe/componente *j*
- **1**: dipendenza diretta
- **2+**: dipendenza indiretta tramite una o più classi/componenti intermedi

La matrice consente di evidenziare la profondità effettiva di propagazione delle chiamate tra le componenti centrali del sistema, supportando un'analisi più accurata dell'impatto di eventuali modifiche.

1.1. Obiettivo

Gli obiettivi del documento sono:

1. Condurre un'analisi statica della codebase per analizzare la struttura del sistema CodeSmile, esaminandone la composizione a livello di classi/servizi, al fine di individuare relazioni di dipendenza che influenzano l'impatto delle modifiche. Le analisi condotte si basano su parsing statico del codice Python, supportato da strumenti di elaborazione automatica dei grafi di chiamata.
2. Caratterizzare l'impatto di ciascuna Change Request.
3. Motivare la previsione di impatto tramite evidenze strutturali (call graph/matrice di raggiungibilità), rendendo tracciabile il passaggio da dipendenze osservabili a insiemi di impatto.
4. Valutare la qualità dell'analisi mediante metriche di accuratezza (Precision/Recall) calcolate sul confronto tra l'insieme delle componenti potenzialmente impattate e l'nsieme delle componenti realmente modificate durante l'implementazione, al fine di misurare quanto l'analisi anticipi correttamente i cambiamenti effettivi.

1.2. Descrizione delle Change Request

ID	Titolo	Descrizione breve
CR01	Allineamento dinamico degli import e degli endpoint tra ambiente di esecuzione locale e containerizzato	Automatizza la configurazione tra locale e Docker, eliminando la modifica manuale degli import e degli URL nei servizi.
CR02	Introduzione della generazione di Call Graph nella CLI di CodeSmile	Estende la CLI per generare call graph durante l'analisi statica, arricchendo i report con la struttura delle chiamate.
CR03	Integrazione di call graph interattivo con visualizzazione degli smell rilevati tramite analisi statica	Aggiunge nella webapp la visualizzazione interattiva del call graph.

1.3. Definizioni e acronimi

- **SIS (Starting Impact Set):** Insieme delle componenti direttamente (o inizialmente) coinvolte dalla Change Request.
- **CIS (Candidate Impact Set):** Insieme delle componenti potenzialmente colpite a livello indiretto.
- **AIS (Actual Impact Set):** Insieme delle componenti realmente modificate durante l'implementazione.
- **FPIS (False Positive Impact Set):** Componenti inizialmente considerate a rischio ma che non subiscono modifiche.
- **DIS (Discovered Impact Set):** Componenti non previste inizialmente ma risultate impattate in corso d'opera.

2. Change Request CR01

La presente Change Request introduce una normalizzazione della codebase della Web App basata su microservizi, con l'obiettivo di rendere l'esecuzione agnostica rispetto all'ambiente (locale vs Docker). Nella situazione attuale, l'esecuzione in locale e l'esecuzione containerizzata richiedono modifiche manuali al codice, in particolare, sono presenti commenti per gestire import alternativi (es. import assoluti "webapp…" vs import relativi "app…") e configurazioni hardcoded degli endpoint (es. localhost in locale vs nomi dei servizi in rete Docker).

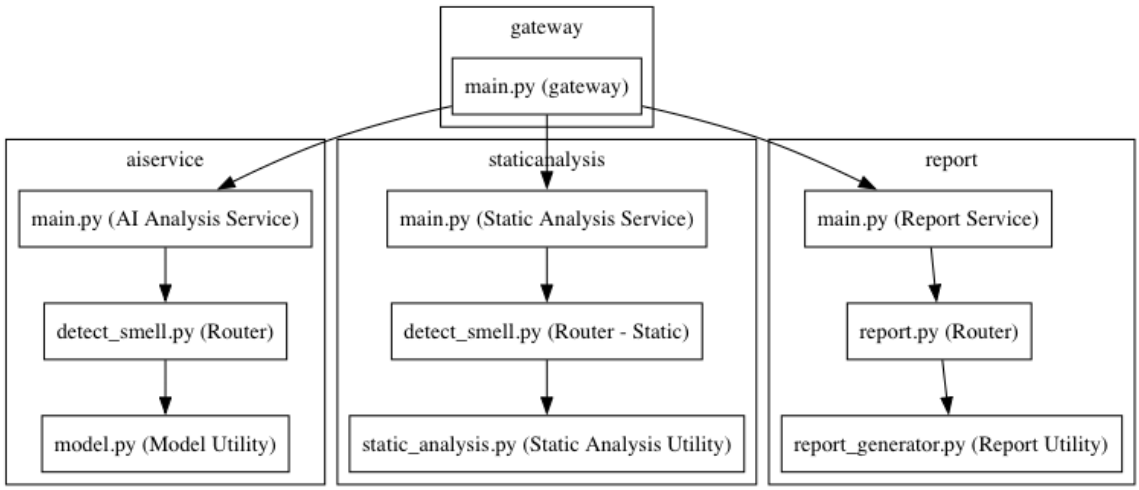
Situazione attuale: Import gestiti tramite blocchi commentati ed alternative manuali per passare tra esecuzione locale e Docker. URL dei servizi hardcoded nel gateway e nei servizi, con valori differenti tra locale e container. La struttura dei container non replica fedelmente la struttura dei package del progetto in locale, generando incongruenze nell'import.

Situazione desiderata: Un'unica codebase pulita, senza dover commentare e decommentare manualmente. Docker configurato per replicare la stessa struttura dei package del progetto locale adattando l'infrastruttura al codice, non il contrario.

2.1. Analisi statica

Il seguente call graph architetturale rappresenta la Web App a microservizi: il gateway invoca i tre servizi (AI Analysis, Static Analysis, Report). Ogni servizio è strutturato su tre livelli:

- main.py (entrypoint del servizio)
- router (gestione endpoint)
- utility (logica di supporto richiamata dal router)



La matrice seguente rappresenta la raggiungibilità tra i componenti principali già mostrate nel call graph.

	gateway_main	ai_main	ai_router	ai_utility	static_main	static_router	static_utility	report_main	report_router	report_utility
gateway_main	0	1	2	3	1	2	3	1	2	3
ai_main	-	0	1	2	-	-	-	-	-	-
ai_router	-	-	0	1	-	-	-	-	-	-
ai_utility	-	-	-	0	-	-	-	-	-	-
static_main	-	-	-	-	0	1	2	-	-	-
static_router	-	-	-	-	-	0	1	-	-	-
static_utility	-	-	-	-	-	-	0	-	-	-
report_main	-	-	-	-	-	-	-	0	1	2
report_router	-	-	-	-	-	-	-	-	0	1
report_utility	-	-	-	-	-	-	-	-	-	0

2.2. Impact Analysis

2.2.1. SIS (Start Impact Set)

I componenti direttamente coinvolti in questa CR sono:

- commenti sugli import (locale vs Docker),
- URL hardcoded o configurazioni diverse tra ambienti,
- configurazioni Docker necessarie per allineare l'ambiente containerizzato alla struttura locale del progetto.

I file individuati e poi effettivamente coinvolti sono:

- **Codice applicativo:**
  1. webapp/gateway/main.py
  2. webapp/services/aiservice/app/main.py
  3. webapp/services/aiservice/app/routers/detect_smell.py
  4. webapp/services/aiservice/app/utils/model.py
  5. webapp/services/staticanalysis/app/main.py
  6. webapp/services/staticanalysis/app/routers/detect_smell.py
  7. webapp/services/staticanalysis/app/utils/static_analysis.py
  8. webapp/services/report/app/main.py
  9. webapp/services/report/app/routers/report.py
- **Infrastruttura Docker:**
  10. docker-compose.yml
  11. webapp/gateway/Dockerfile
  12. webapp/services/aiservice/Dockerfile
  13. webapp/services/staticanalysis/Dockerfile
  14. webapp/services/report/Dockerfile

SIS = {14 file sopra elencati}

2.2.2. CIS (Candidate Impact Set)

Non sono previsti impatti indiretti su altri componenti: **CIS** = SIS

2.3. Implementazione della Change Request

2.3.1. AIS (Actual Impact Set)

L'implementazione della CR è stata completata tramite una strategia chiave, ossia, adattare Docker al codice, non il codice a Docker, in particolare:

- **Standardizzazione degli import:** Rimossi i blocchi commentati. Uniformati gli import in modo che siano coerenti e stabili (import assoluti basati sulla root del progetto, es. from webapp.services...).
- **Configurazione URL dinamica:** Nel gateway, gli URL dei servizi non sono più gestiti tramite doppie definizioni commentate. Gli endpoint vengono letti da variabili d'ambiente quando disponibili con Docker, con default per locale.
- **Build context unificato:** Aggiornato il file docker-compose.yml per usare come build context la root del repository, in modo che l'immagine Docker abbia visibilità completa dei package.
- **Replica della struttura dei package nei container:** Aggiornati i Dockerfile per copiare il codice in un percorso come la struttura locale (es. /app/webapp/...).

I file effettivamente modificati coincidono con quanto previsto nel SIS:

AIS = SIS

2.3.2. FPIS (False Positive Impact Set)

Non sono stati riscontrati file inizialmente previsti che poi non hanno richiesto modifica. Tutte le modifiche pianificate erano necessarie:

FPIS = { ∅ }

2.3.3. DIS (Discovered Impact Set)

Durante l'implementazione non sono emersi componenti logici non previsti:

DIS = { ∅ }

2.4. Metriche

Per valutare la qualità dell'Impact Analysis consideriamo le metriche:

$$Precision = \frac{|CIS \cap AIS|}{|CIS|} = \frac{14}{14} = 1{,}00$$

$$Recall = \frac{|CIS \cap AIS|}{|AIS|} = \frac{14}{14} = 1{,}00$$

L'analisi si è rivelata pienamente accurata, individuando esattamente tutti i componenti necessari all'allineamento tra locale e Docker, senza falsi positivi e senza componenti scoperti in corso d'opera.

3. Change Request CR02

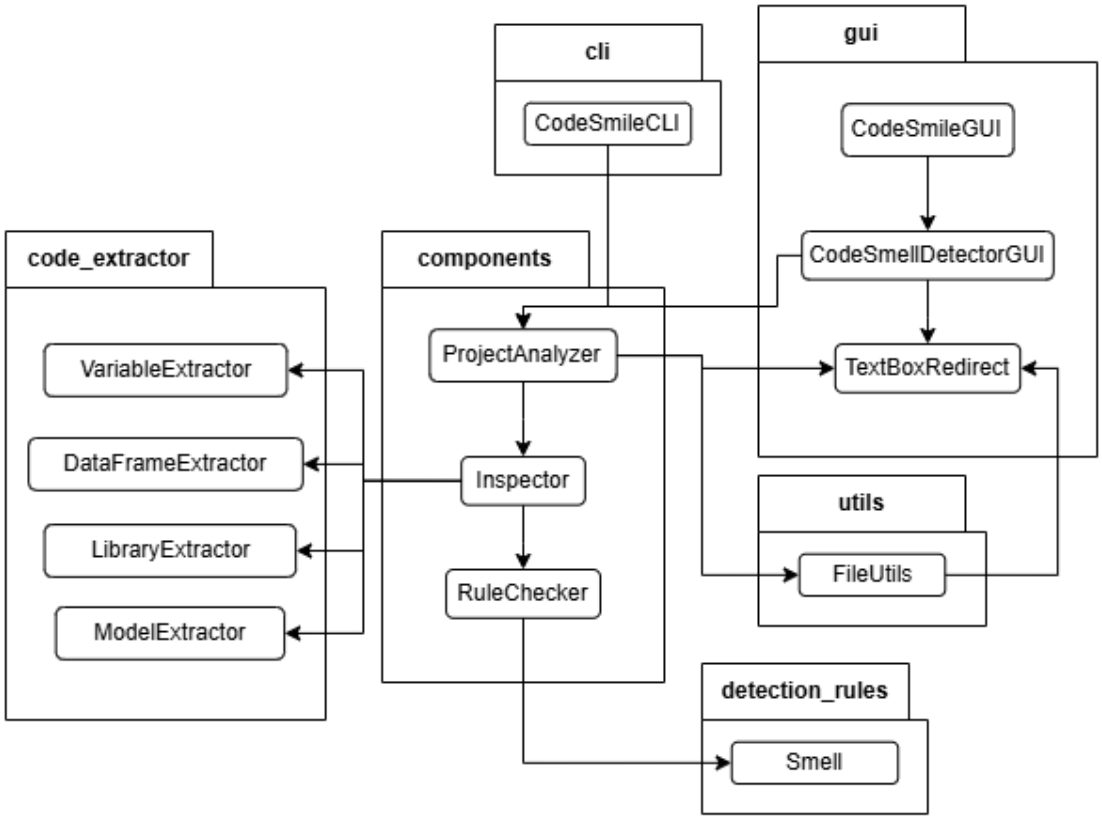
La presente Change Request propone l'estensione della Command Line Interface (CLI) di CodeSmile per supportare la generazione automatica di call graph durante l'analisi dei progetti. La modifica introduce un nuovo comando che attiva una pipeline dedicata alla costruzione dei call graph per il progetto o modulo di input, da integrare nel report finale al fine di fornire una visione più completa della qualità del codice.

Situazione attuale: La CLI di CodeSmile supporta esclusivamente flag per l'analisi degli smell e avvia unicamente il flusso di generazione del report CSV finale. Non sono previste funzionalità per la costruzione o visualizzazione dei call graph, né strutture dati che rappresentino le relazioni di chiamata tra funzioni, limitando l'analisi a un livello puramente descrittivo.

Situazione desiderata: La modifica prevede l'integrazione di una nuova funzionalità nella CLI per la generazione automatica dei call graph per ciascun progetto o modulo analizzato, migliorando la comprensione del design del sistema e della propagazione degli smell a supporto delle attività di refactoring.

3.1. Analisi statica

Il seguente Call Graph illustra le relazioni di chiamata tra le principali classi che costituiscono lo strumento CodeSmile, organizzate per package funzionale. Sono incluse esclusivamente le classi responsabili del coordinamento o dell'elaborazione del processo di analisi dei code smell. Per migliorare la leggibilità, le classi derivate da *Smell* sono state intenzionalmente omesse, mantenendo unicamente il livello di astrazione.



La matrice seguente rappresenta la raggiungibilità tra le principali classi del sistema, già mostrate nel call graph.

	CodeSmellDetectorGUI	CodeSmileCLI	CodeSmileGUI	DataFrameExtractor	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
LibraryExtractor	-	-	-	-	-	-
ModelExtractor	-	-	-	-	-	-
ProjectAnalyzer	-	-	-	2	1	1
RuleChecker	-	-	-	-	-	-
Smell	-	-	-	-	-	-
TextBoxRedirect	-	-	-	-	-	-
VariableExtractor	-	-	-	-	-	-

	LibraryExtractor	ModelExtractor	ProjectAnalyzer	RuleChecker	Smell	TextBoxRedirect	VariableExtractor
CodeSmellDetectorGUI	3	3	1	3	4	1	3
CodeSmileCLI	3	3	1	3	4	2	3
CodeSmileGUI	4	4	2	4	5	2	4
DataFrameExtractor	-	-	-	-	-	-	-
FileUtils	-	-	-	-	-	1	-
Inspector	1	1	-	1	2	-	1
LibraryExtractor	0	-	-	-	-	-	-
ModelExtractor	-	0	-	-	-	-	-
ProjectAnalyzer	2	2	0	2	3	1	2
RuleChecker	-	-	-	0	1	-	-
Smell	-	-	-	-	0	-	-
TextBoxRedirect	-	-	-	-	-	0	-
VariableExtractor	-	-	-	-	-	-	0

3.2. Impact Analysis

3.2.1. SIS (Start Impact Set)

Le classi e i file direttamente coinvolti in questa CR sono:

- **cli/cli_runner.py:** bisogna aggiungere la funzionalità che permette di generare il call graph attraverso un flag.
- **components/project_analyzer.py:** tale modulo gestisce l'analisi dei progetti forniti in input e quindi "orchestra" anche la generazione del call graph.
- **components/inspector.py:** modulo principale che costruisce l'AST per l'identificazione degli smell, che verrà sfruttato anche per la generazione del call graph.

SIS = {cli/cli_runner.py,
components/project_analyzer.py,
components/inspector.py}

3.2.2. CIS (Candidate Impact Set)

Non sono previsti impatti indiretti su altri componenti: **CIS** = SIS

3.3. Implementazione della Change Request

3.3.1. AIS (Actual Impact Set)

L'implementazione della CR è stata completata. I file effettivamente modificati sono:

AIS = {cli/cli_runner.py,
components/call_graph_generator.py,
components/project_analyzer.py}

3.3.2. FPIS (False Positive Impact Set)

L'Inspector era stato inizialmente considerato come componente impattato, ma ciò non è avvenuto, in quanto la logica di creazione del call graph è stata isolata nel nuovo componente:

FPIS = {components/inspector.py}

3.3.3. DIS (Discovered Impact Set)

L'unico file non previsto è il nuovo modulo che separa nettamente l'analisi strutturale (Call Graph) dalla rilevazione degli smell (Inspector):

DIS = {components/call_graph_generator.py}

3.4. Metriche

Per valutare la qualità dell'Impact Analysis consideriamo le metriche:

$$Precision = \frac{|CIS \cap AIS|}{|CIS|} = \frac{2}{3} \cong 0{,}67$$

$$Recall = \frac{|CIS \cap AIS|}{|AIS|} = \frac{2}{3} \cong 0{,}67$$

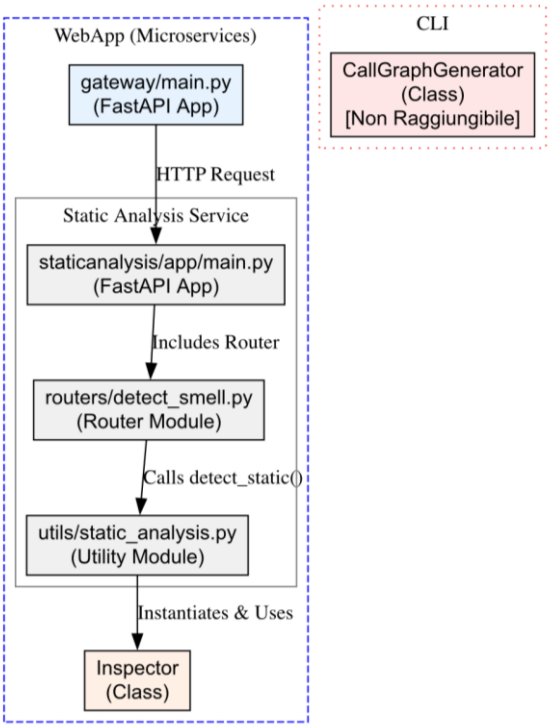
L'analisi si è rivelata discretamente precisa, prevedendo un impatto poco esteso, ma che include comunque un falso positivo e la creazione di un nuovo file.

4. Change Request CR03

La presente Change Request mira ad arricchire l'esperienza utente nella Web App del tool CodeSmile integrando una visualizzazione grafica e interattiva delle dipendenze del codice analizzato, ossia, mostrando un Call Graph ed eventuali code smells presenti.
Situazione attuale: La Web App permette l'upload di singoli file o snippet di codice e restituisce un elenco testuale di code smell rilevati tramite l'Inspector. Esiste un componente CallGraphGenerator nella codebase, ma è isolato e utilizzato esclusivamente dalla CLI. Non esiste alcun collegamento operativo tra la Web App e questo generatore.
Situazione desiderata: Integrazione di una nuova pagina "Call Graph" nella Web App che utilizzi il componente esistente (con le dovute estensioni) per generare call graph interattivi ed evidenziando i nodi con la presenza di eventuali code smells, partendo da progetti caricati come ZIP oppure caricando la folder o un singolo file .py.

4.1. Analisi statica

Il seguente Call Graph architetturale rappresenta lo stato del sistema prima dell'intervento. Si nota chiaramente come il componente *CallGraphGenerator* sia presente nel progetto (CLI) ma tecnicamente irraggiungibile dai servizi della Web App. Le frecce rappresentano dipendenze operative tra componenti (HTTP per i microservizi, chiamate di funzione dirette e uso di classi), non un Call Graph a livello di singola funzione, questa scelta è stata guidata per migliorarne la leggibilità, pur mantenendo unicamente il livello di astrazione.



La matrice seguente rappresenta la raggiungibilità tra le principali classi del sistema, già mostrate nel call graph.

	gateway_main	static_main	static_router	static_analysis_utility	inspector	call_graph_generator
gateway_main	0	1	2	3	4	-
static_main	-	0	1	2	3	-
static_router	-	-	0	1	2	-
static_analysis_utility	-	-	-	0	1	-
inspector	-	-	-	-	0	-
call_graph_generator	-	-	-	-	-	0

4.2. Impact Analysis

4.2.1. SIS (Start Impact Set)

I componenti direttamente coinvolti in questa CR, identificati inizialmente, sono:

- **Frontend:**
  - **webapp/app/call-graph/page.tsx:** nuova pagina per la visualizzazione del Call Graph.
  - **webapp/app/page.tsx:** home page.
  - **webapp/utils/api.ts:** utility API.
- **Backend Services:**
  - **webapp/gateway/main.py:** per inoltro dei payload
  - **webapp/services/staticanalysis/app/routers/detect_smell.py:** endpoint
  - **webapp/services/staticanalysis/app/utils/static_analysis.py:** logica e analisi file .py
  - **webapp/services/staticanalysis/app/schemas/requests.py:** validazione
  - **webapp/services/staticanalysis/app/schemas/responses.py:** formattazione output

SIS = {webapp/app/call-graph/page.tsx, webapp/app/page.tsx, webapp/utils/api.ts, 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}

4.2.2. CIS (Candidate Impact Set)

Non sono previsti impatti indiretti su altri servizi (AI Service, Report Service) dato che la funzionalità è isolata nel flusso di analisi statica. Si ipotizza un possibile impatto su **webapp/services/staticanalysis/app/main.py** per le configurazioni globali.

CIS = SIS + {webapp/services/staticanalysis/app/main.py}

4.3. Implementazione della Change Request

4.3.1. AIS (Actual Impact Set)

L'implementazione della CR03 ha richiesto la modifica di tutti i file previsti nel SIS, con le eccezioni descritte di seguito:

AIS = SIS + DIS - FPIS

4.3.2. FPIS (False Positive Impact Set)

webapp/services/staticanalysis/app/main.py: Inizialmente si pensava fosse necessario modificare la configurazione di avvio del servizio per gestire i file temporanei, ma la libreria tempfile di Python ha reso superflua questa modifica di configurazione:

FPIS = {webapp/services/staticanalysis/app/main.py}

4.3.3. DIS (Discovered Impact Set)

Durante l'implementazione sono emersi tre componenti critici non considerati nell'analisi iniziale:

1. **components/call_graph_generator.py:** Sebbene sia un componente condiviso con la CLI, la WebApp necessita di informazioni aggiuntive per l'interfaccia interattiva (es. il contenuto del codice sorgente riga per riga per il popup) che la CLI non richiedeva. È stato necessario estendere la logica di generazione nodi.
2. **webapp/components/HeaderComponent.tsx:** Necessario aggiungere il pulsante di navigazione nella Navbar per rendere accessibile la nuova pagina oltre che dall pulsante presente nell'Home Page.
3. **webapp/package.json:** Emersa la necessità di gestire la compressione ZIP lato client (jszip) per supportare l'upload di cartelle in modo efficiente.

DIS = {components/call_graph_generator.py, webapp/components/HeaderComponent.tsx, webapp/package.json}

4.4. Metriche

Per valutare la qualità dell'Impact Analysis consideriamo le metriche:

$$Precision = \frac{|CIS \cap AIS|}{|CIS|} = \frac{8}{9} \cong 0{,}88$$

$$Recall = \frac{|CIS \cap AIS|}{|AIS|} = \frac{8}{11} \cong 0{,}72$$

L'analisi si è rivelata abbastanza precisa, individuando correttamente i componenti principali coinvolti nella CR03 (Precision ≈ 0,88), ma ha sottostimato l'impatto reale (Recall ≈ 0,72) perché durante l'implementazione sono emerse dipendenze aggiuntive non previste (DIS) e un falso positivo legato alla configurazione del servizio.