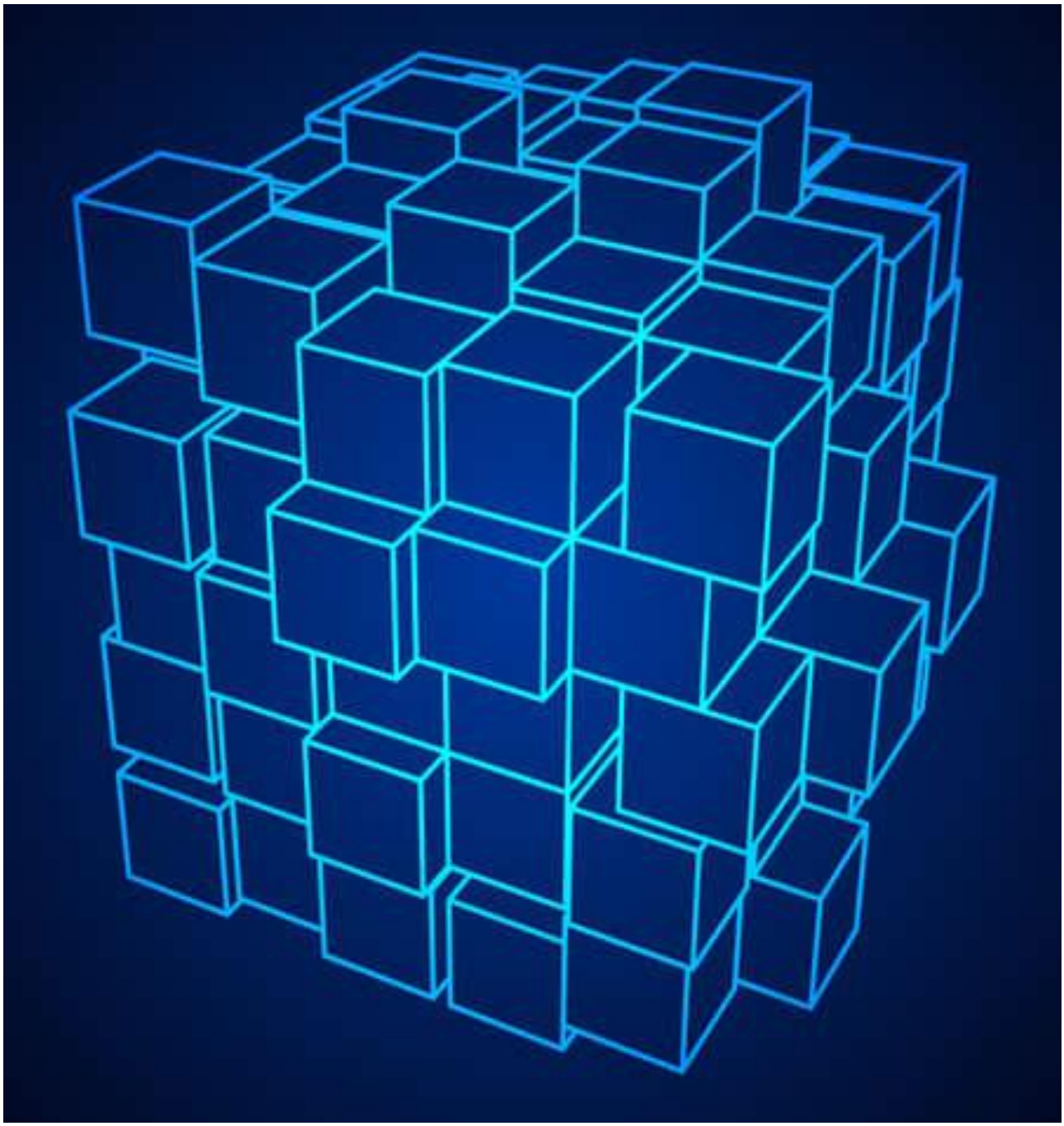


GENCLASS

PROGETTO ENAC – TASK T24-B18
SOFTWARE ARCHITECTURE DESIGN

Alessio Avallone

M63001221

2024/2025

SOMMARIO

0.INFORMAZIONI PROGETTO

Contributore: Alessio Avallone (ales.avallone@studenti.unina.it)

URL: alexgit933/A13: Versione migliorativa sviluppata a partire dal progetto A10-2024 con integrazione del repository A7

La copertina è rappresentativa del concetto di microservizi collegati tra loro.

1.DESCRIZIONE DEL PROGETTO

1.2 Specifiche di progetto

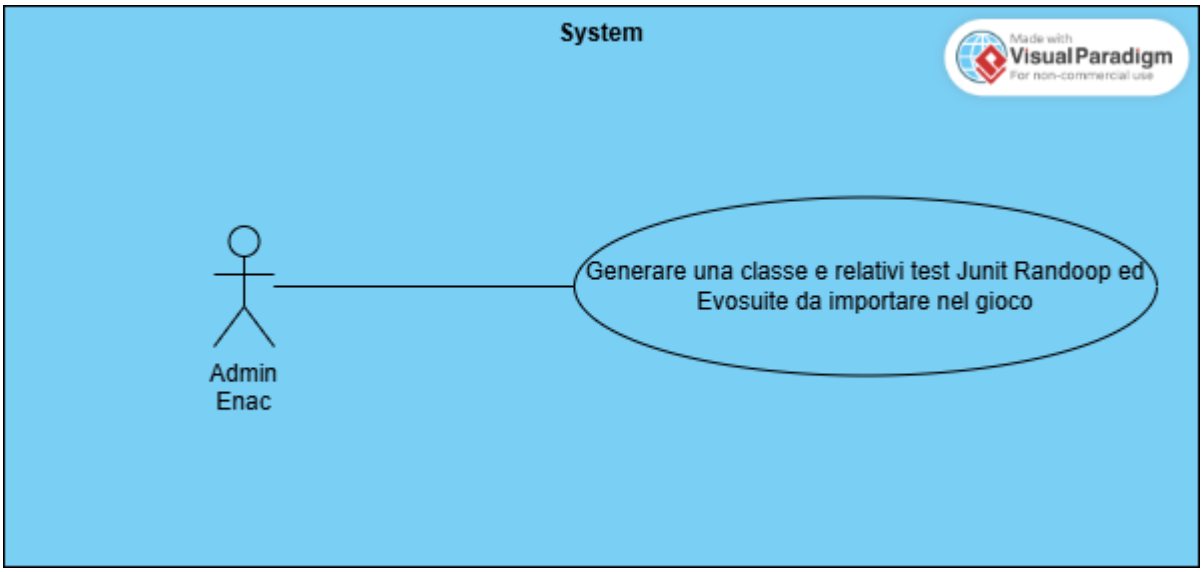
L'endpoint sviluppato permette di generare attraverso un LLM una classe con i relativi test Randoop ed Evosuite, a partire da una richiesta HTTP GET attraverso un architettura REST.

1.3 Processo di sviluppo

Come processo di sviluppo si è scelto un processo di tipo iterativo, articolato in tre iterazioni:

- La prima consisteva nello sviluppare una prima versione che potesse dialogare con il container Docker del LLM.
- La seconda che una volta ottenuta la classe dal modello generasse i test JUnit attraverso Randoop ed Evosuite.
- La terza prevedeva la ricezione e la risposta attraverso un server http REST attraverso API RESTful.

1.4 Diagramma UML (Use Case)



1.5 Scenario

Caso d'uso	Genera classe e Test
Attori	Admin
Descrizione	Genera classe e test Randoop/Evosuite per ENAC.
Precondizione	La cartella generated contente i file temporanei viene per sicurezza cancellata all'avvio di ogni elaborazione.
Sequenza	1.ENAC invia la richiesta HTTP GET 2.Genclass elabora la richiesta 3.Viene inviata ad Ollama il prompt 4.Ollama restituisce la classe 5. Genclass salva la classe e genera il class model 6.Genclass compila la classe (con JDK 8) 7.Genclass avvia attraverso bash Randoop e confeziona lo zip seguendo una precisa struttura 8. Genclass avvia attraverso bash (switchando a JDK 8 per motivi di compatibilità) Evosuite e confeziona lo zip seguendo una precisa struttura 9.Genclass restituisce attraverso http class model classe e i due zip contenente le classi JUnit di Randoop ed Evosuite.
Postcondizione	Il sistema ritorna allo stato iniziale in attesa di una nuova richiesta. La classe Generator dovrebbe venir deallocata dalla JVM.
Sequenza alternativa	Se si verifica un errore (segnalato da eccezioni) dal punto 3 al punto 8 automaticamente il sistema ritorna appunto al punto 3.

2.ANALISI DEI REQUISITI

2.1 Requisiti informali

Il sistema è strutturato in questo modo : l'applicazione principale Genclass riceve la GET da genera classe dal progetto ENAC (sezione admin), prepara e contatta il server OLLAMA attraverso una GET dove risiede il modello LLM (in questo caso qwen2.5-coder (Alibaba, tra i migliori offline del 2024), nella versione 0.5B di parametri) e chiede di generare una classe da un oggetto secondo il prompt "Generate a complete and correct java class from a thing".

A sua volta il modello elabora il prompt e ritorna la classe generata compilando il relativo model come già visto nella documentazione principale di ENAC.

Successivamente vengono chiamati attraverso bash (costituendo di fatto una sorta di pipeline) in questo modo:

- Javac (che compilerà la classe, in modalità JDK 8)
- Randoop
- Evosuite

Le relative classi e materiale vengo da GC usati ricostituendo la struttura dei file di test di Randoop ed evosuite seguendo i file di esempio pubblicati sul repository Github.

In seguito come risposta ad ENAC verranno restituiti da Genclass attraverso risposta HTTP dalla richiesta principale il class model, la classe generata e i due archivi zip post-processati da Genclass contente le classi JUnit più il relativo materiale.

Attualmente a causa di incompatibilità lato ENAC(un generico errore 500), è possibile testare con successo e come stabilito da requisiti funzionali il sistema solo attraverso tool tipo Postman facendo una GET su localhost:8002/generaClasse.

2.2 Requisiti Formali

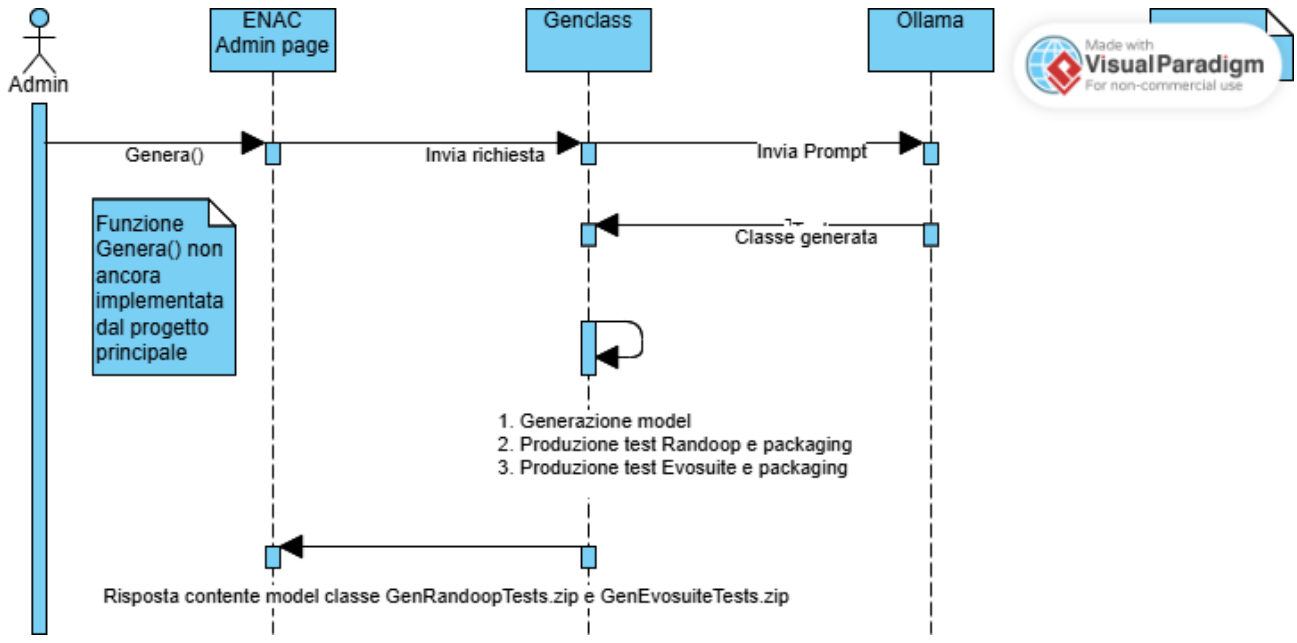
Da colloquio, è stato possibile estrarre questo requisito funzionale:

- Genera classe :Il sistema deve prendere in ingresso una richiesta di classe e generare a sua volta classe, class model e relative classi test JUnit (in seguito Randoop ed Evosuite).

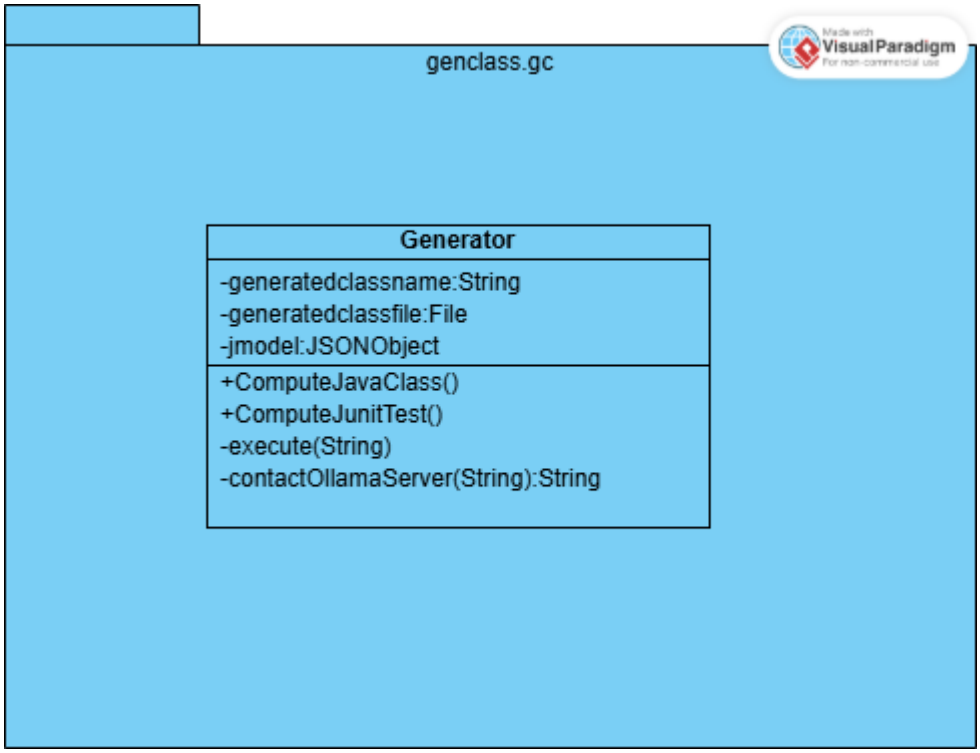
Ed i seguenti requisiti non funzionali :

- Modularità
- Disponibilità
- Velocità di esecuzione
- Modularità
- Interoperabilità
- UX per utenti finali
- Manutenibilità

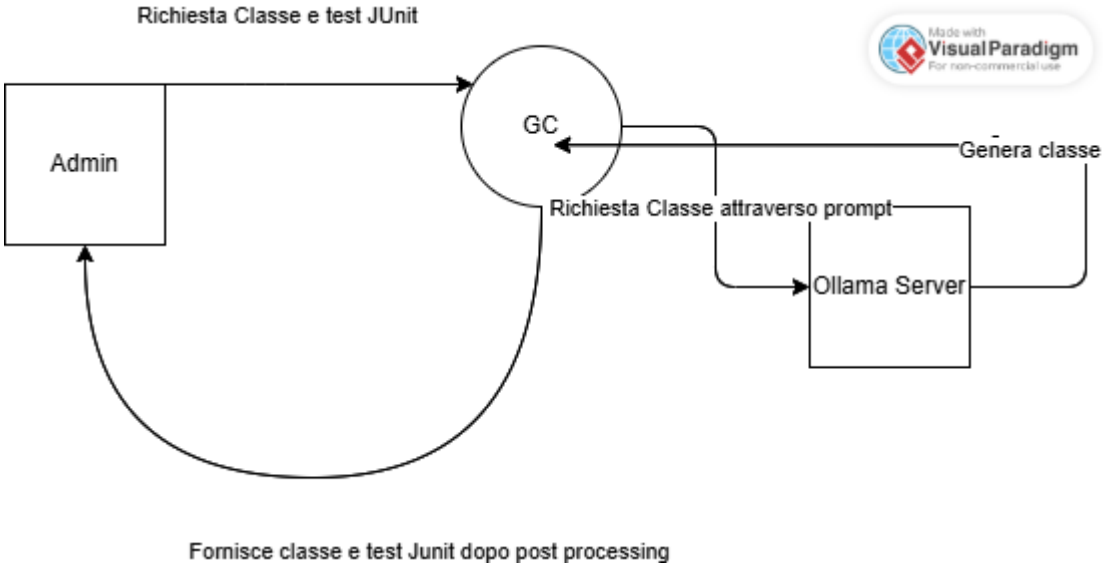
2.3 Sequence Diagram Genera classe



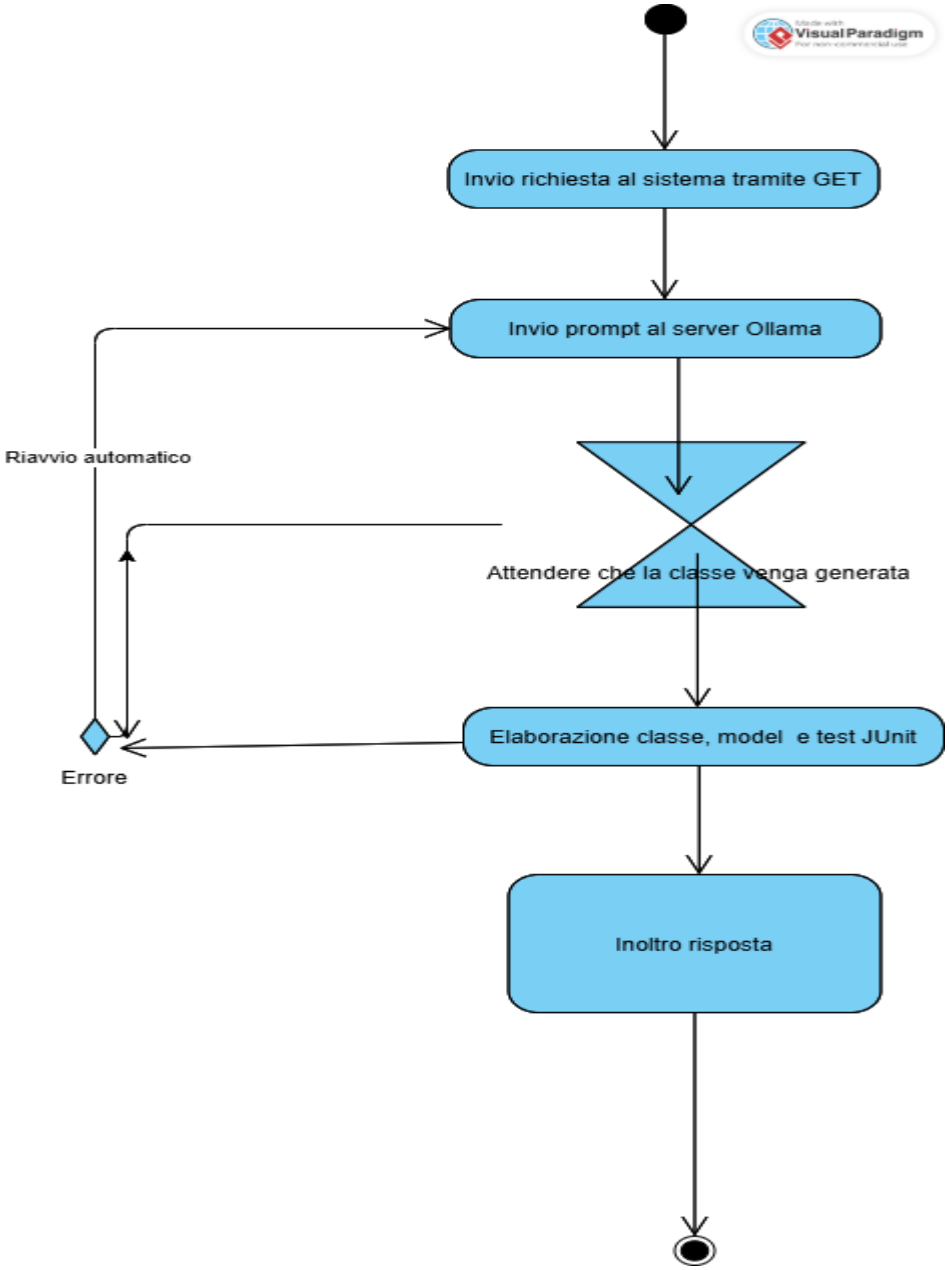
2.4 Class Diagram Genclass



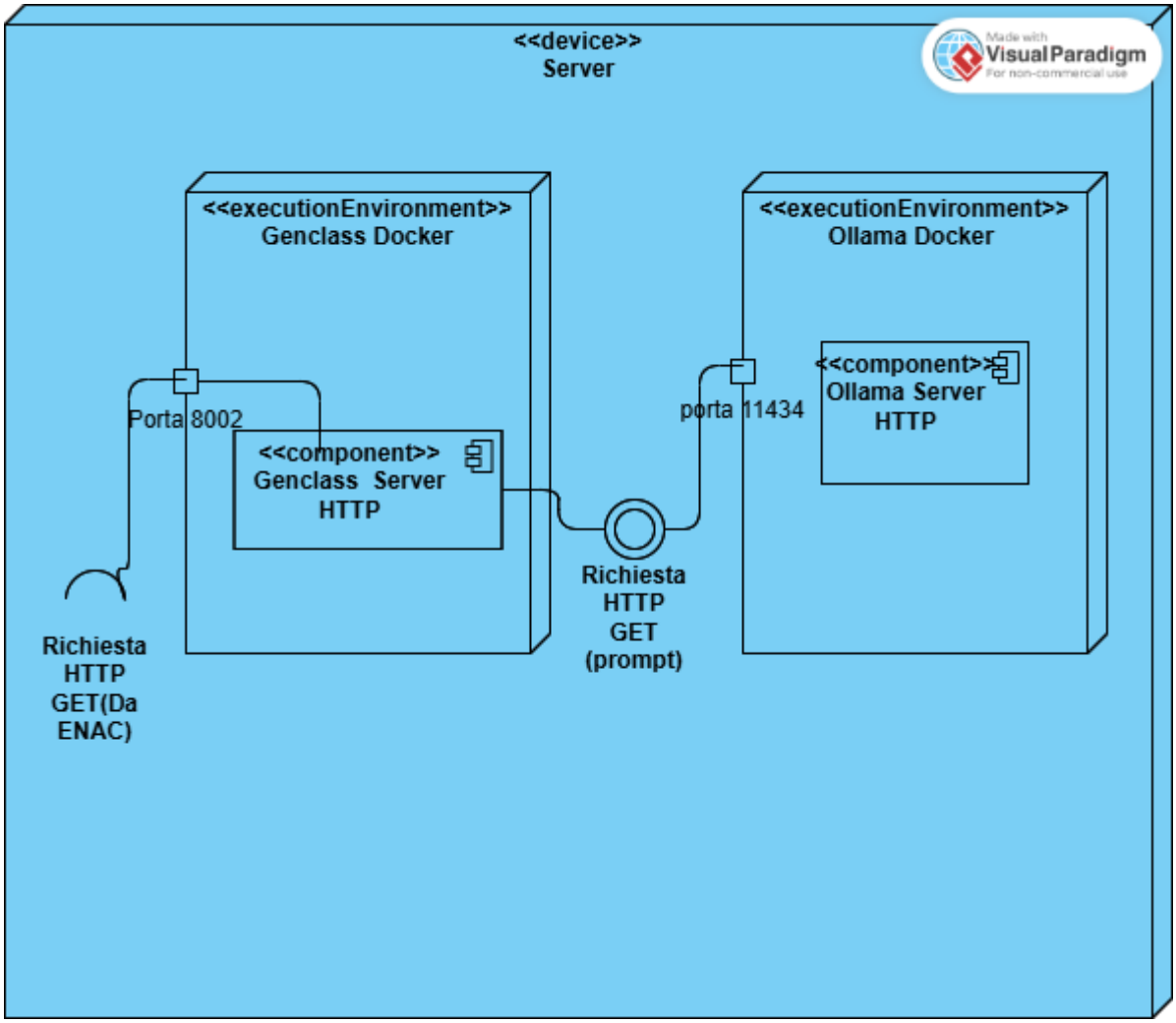
2.5 Context diagram Genclass



2.6 Activity Diagram Genera classe



2.7 Deploy diagram Genclass



3.ANALISI DELL'IMPATTO DEI REQUISITI SUL PROGETTO ENAC

3.1 Analisi

Attraverso lo sviluppo dei requisiti il sistema sarà in grado di espandere potenzialmente all'infinito le dinamiche di gioco, attraverso la generazione delle classi usando un LLM, seguendo il nascente e sempre più apprezzato trend sperimentale della gamification, la generazione procedurale del materiale di gioco.

4.PROGETTAZIONE DELLA SOLUZIONE

4.1 Descrizione

L'app è progettata in Java usando JDK 21 (si è scelto di usare la versione 21 poiché LTS) ed eseguita in un container Docker in ambiente Ubuntu 24.04 (anch'essa LTS). È strutturata in tre classi principali, Main, Generator e Utils.

La classe Main, entry point, istanzia il server HTTP in ascolto su tutte le interfacce sull'indirizzo /generaClasse su porta 8002 ed inoltre gestisce le richieste GET attraverso l'apposito handler, per poi inviare la risposta attraverso JSON e dati binari.

La classe Generator, gestisce l'elaborazione principale della class model, della classe e dei relativi test Randoop ed Evosuite che verranno poi presi da Main per inviare la risposta.

Utils è una classe di supporto, inserita per promuovere la coesione e la modularità della classe Generator sia a livello di codice che di UML.

In particolar modo dopo aver accolto la richiesta GET ,Main istanzia la classe Generator richiamando i due metodi ComputeJavaClass() ed ComputeJunitTest().Caratteristica interessante è il fatto che nel caso di eccezioni catturate all'interno del blocco try catch relative a errori nei due metodi, viene richiamato di nuovo la catena dei dei metodi fino al successo dell'operazione (esito= true, come si cede dal codice) per massimo 50 volte (limite inserito in quando in assenza di questo stranamente crashava il Docker Engine, ma non se eseguito al di fuori di esso).

Nella classe Generator il primo metodo contatta il server Ollama usando il metodo contactOllamaServer (String prompt) attraverso una richiesta HTTP all'indirizzo /api/generate su porta 11434 inviando i dati di configurazione (modello e tipologia di stream, quest'ultimo impostato su off poichè vogliamo i risultati della generazione in una sola volta e non carattere per carattere) e il prompt "Generate a complete and correct java class from a thing".

Ci sono diverse considerazioni da fare: prima cosa è stato scelto il modello tra i migliori per coding locale, ma nella declinazione da 0.5B(Billion, Miliardi) di parametri per problemi di prestazioni.

Nulla vieta di cambiare il numero di parametri scegliendo una versione consona dalla pagina del modello qwen2.5-coder sul sito di Ollama.

Si nota che il modello da 0.5B a seguito di svariati test raramente sbaglia e genera classe Java non conformi, cosa che il modello da 7B sembra non fare. Questo è dovuto a una rete neurale poco densa dove anche meccanismi come l'attention tipica dei LLM transformer sembra venir meno. In ogni caso, javac andrà in errore e solleverà un eccezione dove il programma ripartirà dal primo metodo come detto sopra.

Successivamente dopo aver ricevuto la risposta, tramite regex viene estrapolata solo la classe ignorando i commenti del LLM(che se anche gli viene chiesto di non commentare, commenta lo stesso per il problema spiegato sopra) .La classe viene salvata nella cartella generated e compilato il class model pienamente conforme al progetto ENAC.

Nel secondo metodo ComputeJunitTest() viene eseguito in ordine javac(Java compiler) , Randoop ed Evosuite. In particolar modo javac compila solo classi compatibile con JDK 8 in quanto Evosuite non è in grado di elaborare classi di JDK superiori. Randoop ed Evosuite sono eseguiti in processi separati come programmi esterni.

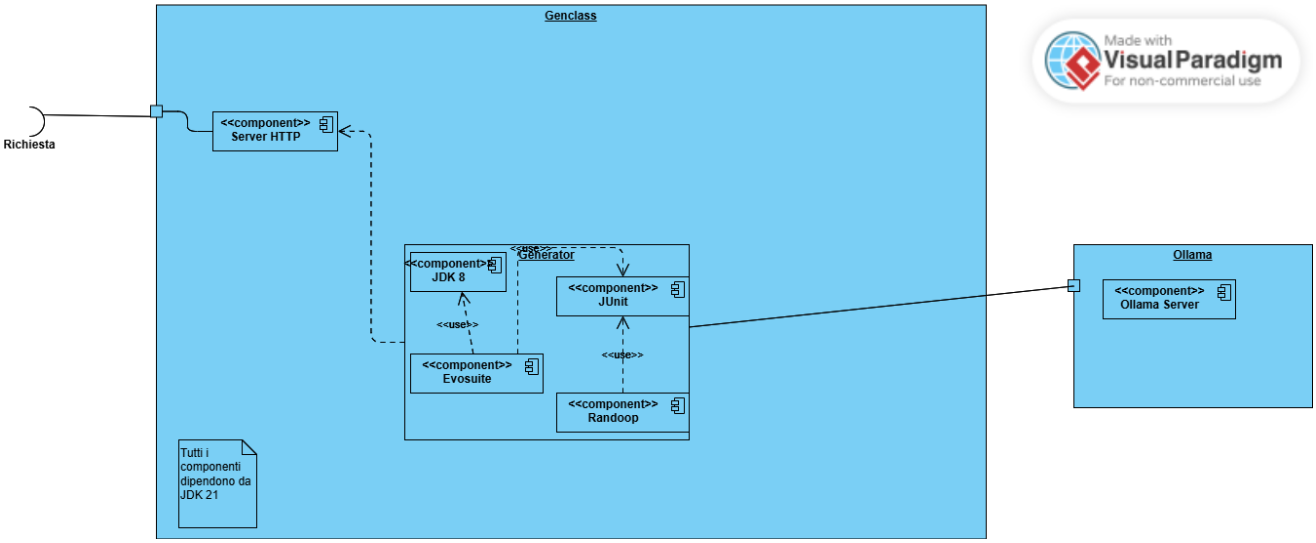
Da notare che Evosuite non è compatibile con JDK21, pertanto attraverso bash si è dovuto provvedere a richiamare uno script che setta le variabili di ambiente JAVA_HOME e PATH corrispondenti a un installazione JDK 8(lo script è prodotto stesso da Genclass ed viene salvato come file temporaneo prima di essere eseguito).La struttura dei file zip segue quella dei file zip presenti del progetto,dopo essere stata ricostruita all'interno del metodo.

Infine, avendo a disposizione class model, classe generata e i due file zip, questi vengono inviati come risposta al client ENAC, portando così a termine l'esecuzione dell' handler, pronto per una nuova richiesta.

Note :

- In caso di errore viene sempre mostrato lo stacktrace. In caso di errori non gestiti dalle eccezioni il server restituisce anche un errore 500.
- Per alcune dipendenze è stato usato come gestore di dipendenze Maven, per problemi di sicurezza invece alcune dipendenze sono gestite manualmente.
- Randoop ed Evosuite sono inseriti all'interno del Docker all'atto del building nella direttiva COPY.
- La porta 8002 è stata scelta casualmente, la porta 11434 del server Ollama è quella di default (nel caso di un sistema di produzione sensibile potrebbe costituire una bad practice, ma non è il caso del sistema in questione. Nonostante ciò nelle risposte alla GET non si fa riferimento a componenti interni, versioni in particolare o protocolli in generale).
- Per seguire il principio dell'incapsulazione sono stati inserite funzioni GET solo dei dati membro strettamente necessari, ed esposte con scope public le classi necessarie.
- Per quanto possibile sono state utilizzati componenti Long Term Support.
- Il codice è stato commentato il più chiaramente possibile usando anche Javadoc.
- JSON resta probabilmente la scelta migliore per la massima interoperabilità (fondamentale in un architettura a microservizi) ed human readibility.

4.2 Component diagram Genclass



4.3 Rotta REST API

Il task aggiunge quindi una nuova API /generaClasse, cosi formalizzata:

Descrizione	Genera una classe attraverso un LLM di ultima generazione, compila il relativo class model e genera attraverso i tool esterni Randoop ed Evosuite le relative classi di test JUnit in formato zip conforme al progetto ENAC. Restituisce il class model in JSON e gli altri tre file come file binari.
URL	http://localhost:8002/generaClasse
Metodo	GET
Risposta	Class model: JSON Classe:binario Test Randoop:binario Test Evosuite:binario
Errori	500 con stacktrace

4.4 Issue corrette

- Nel task T23 la classe Users non compilava correttamente, **IMPEDENDO** il building dell'intero progetto ENAC.
- Nel task T23 era presente un pom.xml malformato.

4.5 Varie

Il container di Ollama con all'interno il modello qwen2.5-coder è stato costruito a partire dal container ufficiale Ollama/Ollama:latest.

Questo permette ad ogni build di avere la versione più recente di Ollama.

5. IMPLEMENTAZIONE E STRUTTURA DEL PROGETTO MODIFICATO

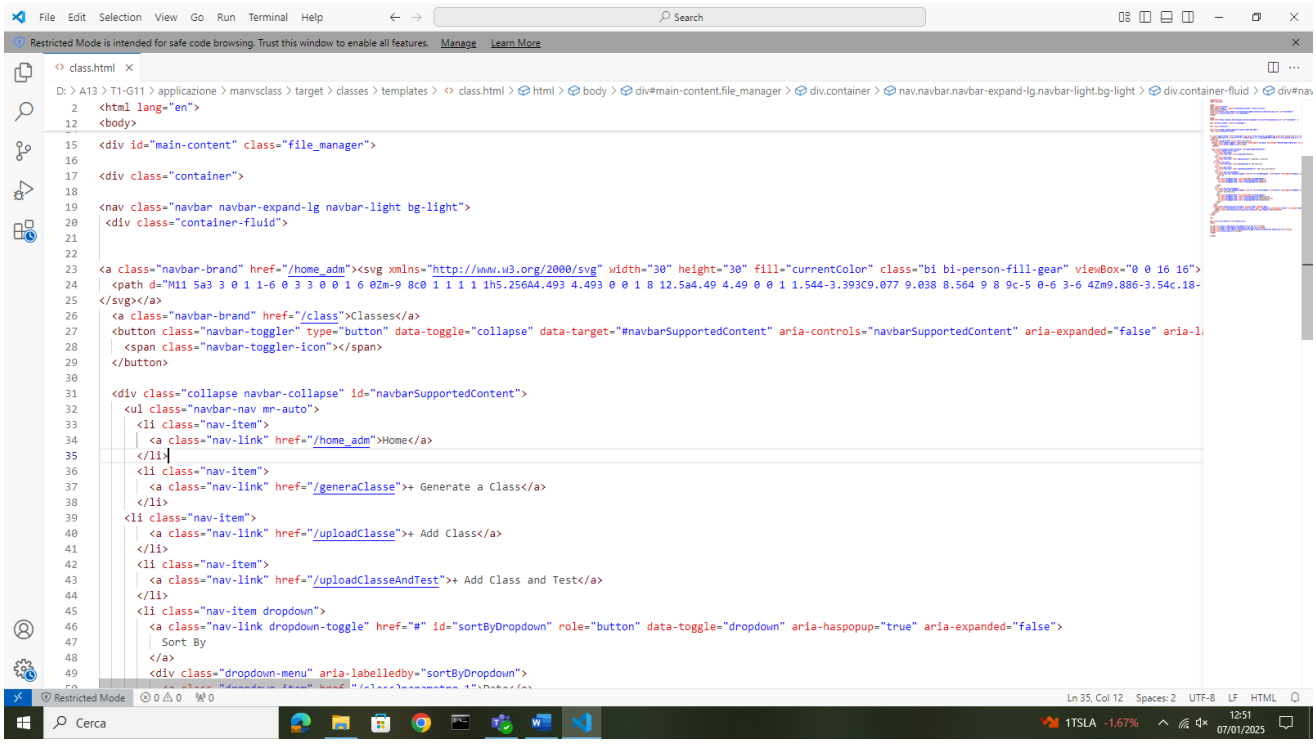
5.1 Lista moduli aggiunti/modificati

Il task T24 è stato aggiunto al progetto (Genclass).

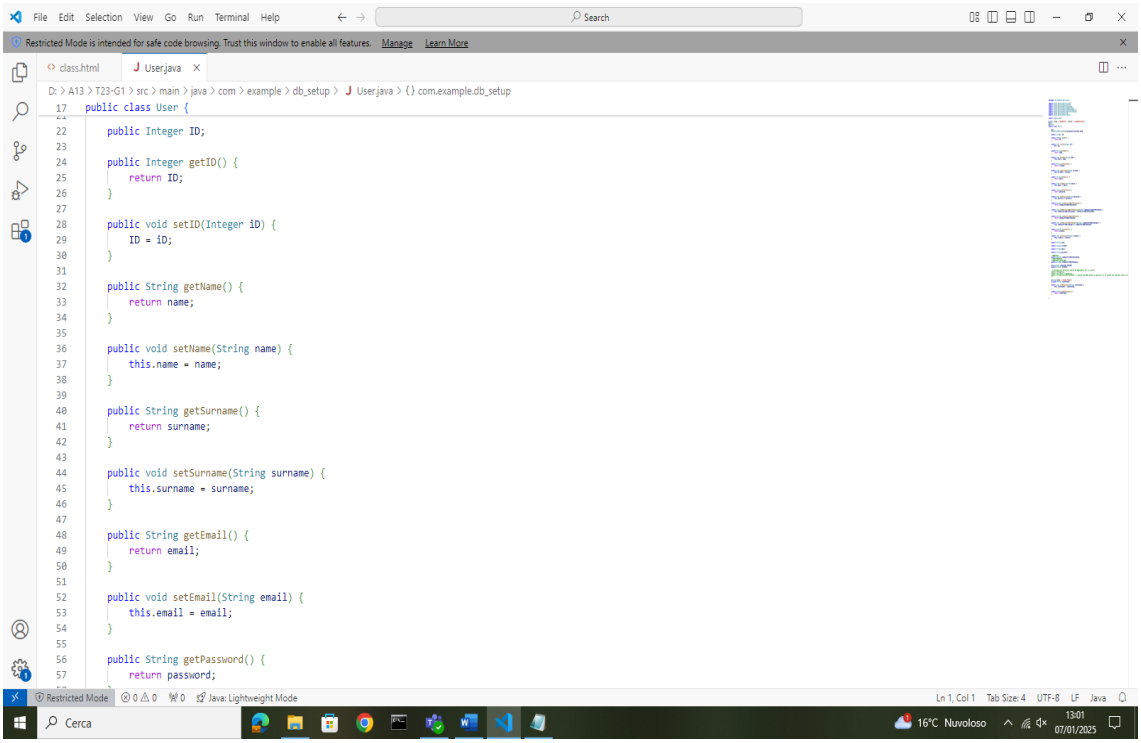
I task T1/T23 sono stati modificati.

5.2 Modifiche ai componenti esistenti

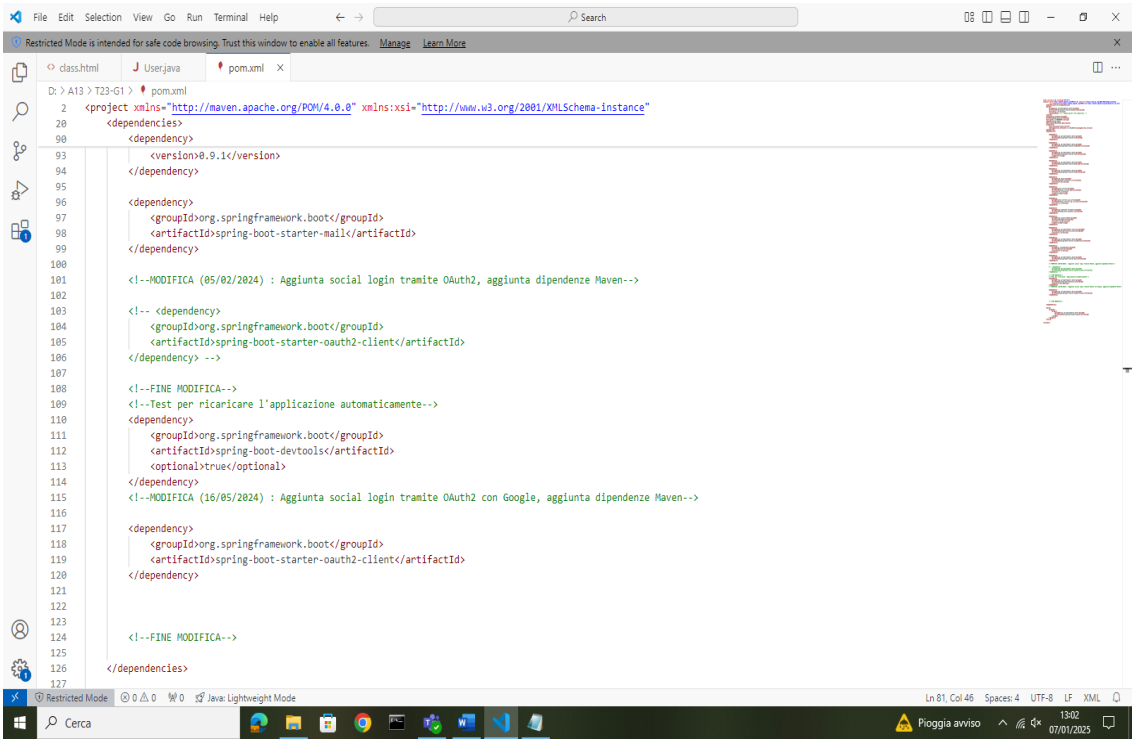
- Nel task T1 è stato aggiunto nel file class.html il pulsante “Generate a class” nella navbar:



- Nel task T23 sono state apportate due **importanti** modifiche:
  - Nella classe User.java mancavo alcune funzioni Getter e Setter, ciò **impediva** il building dell’intero progetto così come prelevato del repository.



- Nel file pom.xml, Maven segnalava l'errore di pom.xml **malformato**, dopo ulteriore analisi si è scoperto che era inserita due volte una dipendenza.



6.TEST EFFETTUATI

6.1 Descrizione

- Test per la funzionalità principale Genera Classe:

Test case ID	Descrizione	Pre-condizioni	Input	Output Attesi	Post-condizioni attese	Output ottenuti	Post-condizioni ottenute	Esito
1	Ollama Server non raggiungibile	Ollama non è raggiungibile dall'endpoint principale	Arresto Ollama	Segnalazione e riavvio automatico	n.d	Segnalazione e riavvio automatico	n.d	PASS
2	Qwen non genera una classe valida	Qwen non genera una classe Java conforme	n.d	Segnalazione e riavvio automatico	n.d	Segnalazione e riavvio automatico	n.d	PASS
3	Randoop ed Evosuite non presenti	I file jar non sono nel path di installazione	Rimozione file in oggetto	Segnalazione e riavvio automatico	n.d	Segnalazione e riavvio automatico	n.d	PASS
4	Errore di binding	IP:Porta in uso sullo stesso host	Esecuzione di due Server Ollama sia in Windows che in WSL contemporaneamente	Segnalazione	Arresto del processo	Segnalazione	Arresto del processo	PASS

Poiché il task non prevede valori in ingresso, ne tantomeno inserimento/modifica/cancellazione valori ed interazioni da parte dell'utente durante la sua esecuzione, potenzialmente non è previsto una lunga casistica di test effettuabili, a meno di bug di rara frequenza da parte della librerie incluse nel progetto o configurazioni particolari di networking.

Inoltre, tuttora e allo stato attuale a causa della natura aleatoria delle reti neurali di deep learning, è sicuramente complesso, talvolta impossibile, testare a fondo sistemi che le utilizzino.

7. SVILUPPI FUTURI

7.1 Sviluppi futuri

In futuro potrebbe essere utile aggiungere due diverse feature :

- Supporto a NVIDIA CUDA: Docker per Ollama supporta nativamente CUDA se abilitato all'atto del run del container attraverso una specifica procedura (non inclusa nel progetto corrente).
  Necessita di NVIDIA CUDA Toolkit installato su un server Linux-based e di una GPU NVIDIA compatibile .
  Si ricorda che nell'ambito delle reti di Deep Learning le soluzioni di GPU computing come CUDA possono migliorare le performance di circa un ordine di grandezza.

- Scelta automatica basata su benchmark del numero di parametri del LLM, all'atto del building del container Ollama, attraverso apposito script.