UNIVERSITÀ DEGLI STUDI DI NAPOLI

FEDERICO II

Software Architecture Design

Corso di Laurea Magistrale in Ingegneria Informatica

Dipartimento di Ingegneria Elettrica e delle tecnologie dell'informazione

GRUPPO 21-TASK8

Emanuele Agostino Messuri M63001547
Teresa Di Dona M63001437
Preziosa Pietroluongo M63001456
Pasquale Carusone M63001459

ANNO ACCADEMICO 2022/2023
Secondo Semestre

Sommario

1. Descrizione del progetto

1.1 Specifiche di progetto

L'applicazione deve offrire la funzionalità di esecuzione del robot Evosuite su una data classe Java. Tale funzionalità riceverà in input un file di testo (classe da testare), dovrà lanciare il generatore di Test, restituendo in output le informazioni relative all'esito dei test del Robot. L'esito dell'esecuzione dovrà essere elaborato in maniera da estrarre da essi le informazioni rilevanti ai fini del gioco (ad esempio la copertura del codice, decisioni, etc.).

1.2 Processo di sviluppo

Sono state adottate metodologie agili al fine di avere i vantaggi di un approccio iterativo ed evolutivo, con la pianificazione di iterazioni brevi ed eventuali feedback. In particolare, si è scelto di seguire il processo di sviluppo Agile Unified Process (AUP), il quale è stato organizzato suddividendo il lavoro di sviluppo software in quattro iterazioni.

Ogni iterazione produce un sistema eseguibile, testato ed integrato con quanto prodotto nelle cinque precedenti iterazioni; comprende attività di analisi dei requisiti, progettazione ed implementazione. Il processo di sviluppo AUP evolve attraverso cicli di "Costruzione-Feedback-Adattamento" e risulta essere molto flessibile: durante ogni iterazione, il team ha aggiornato la documentazione affinché risultasse coerente con il sistema in produzione. I feedback ricevuti hanno influenzato le attività eseguite nelle varie iterazioni.

Il processo di sviluppo è costituito dalle seguenti quattro fasi:

• Ideazione: fase di pianificazione degli obiettivi generali e stima dei costi e dei tempi;

• Elaborazione: fase di implementazione del nucleo dell'architettura del software da realizzare e risoluzione di rischi maggiori;

• Costruzione: fase di implementazione interattiva degli elementi rimanenti, a minor costo;

• Transizione: fase in cui avviene il testing complessivo, il completamento della documentazione ed il rilascio

1.3 Gestione del gruppo

Il team è composto da quattro membri; ogni membro ha lavorato attivamente a tutte le fasi di sviluppo dell'architettura proposta. Alcune attività sono state svolte in coppia, ma al termine di ogni fase il gruppo si è riunito per discutere dei risultati ottenuti e delle modalità di procedimento. Per coordinare tutti i membri, è stata utilizzata la piattaforma **Notion**.

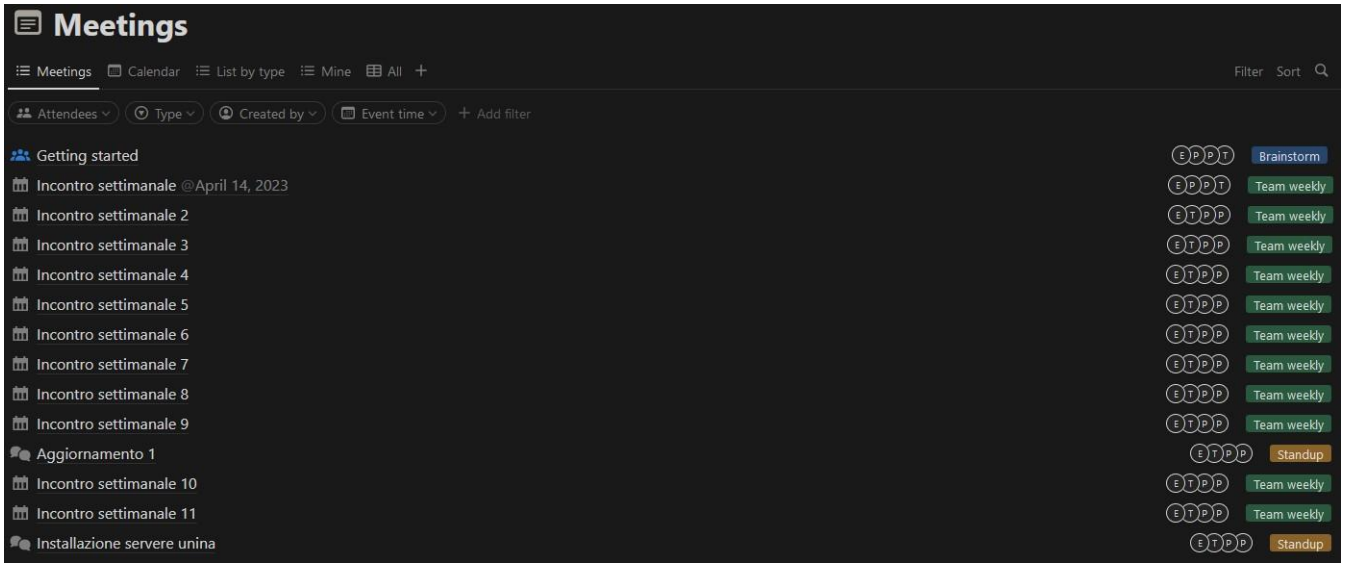


Figura 1: Meeting

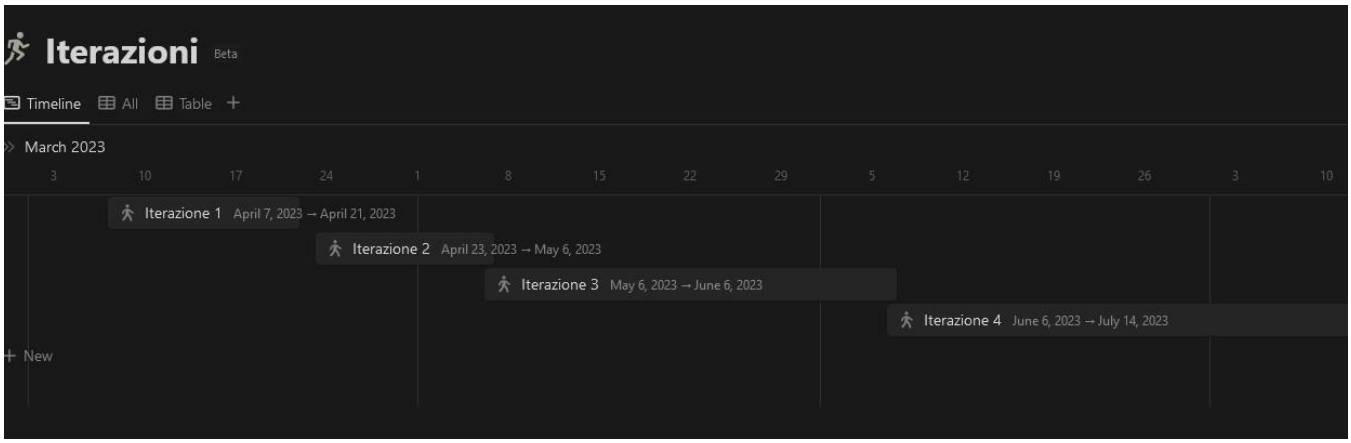
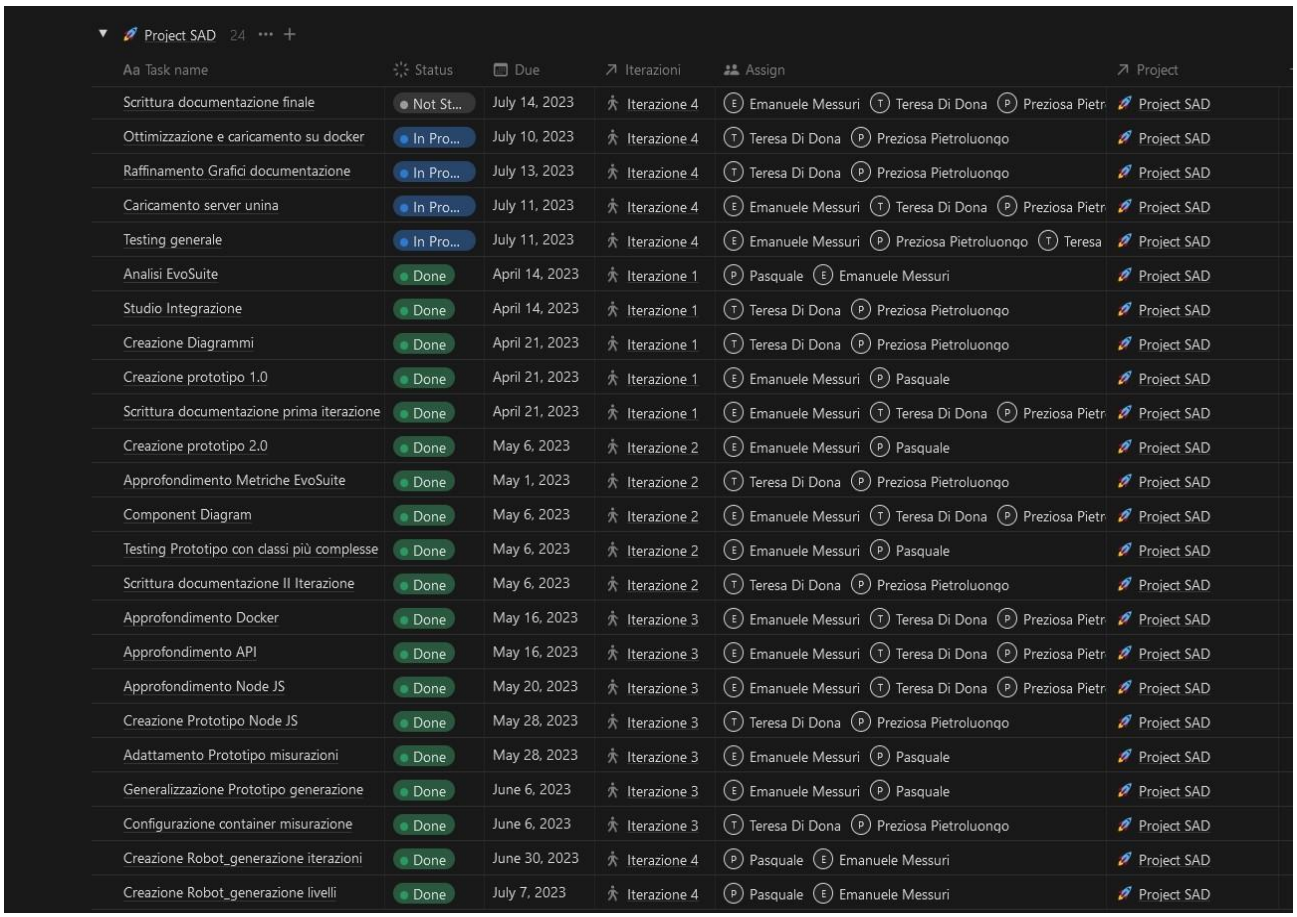


Figura 2: Iterazioni



Project SAD 24

Aa Task name	Status	Due	Iterazioni	Assign	Project
Scrittura documentazione finale	Not St...	July 14, 2023	Iterazione 4	Emanuele Messuri, Teresa Di Dona, Preziosa Pietr	Project SAD
Ottimizzazione e caricamento su docker	In Pro...	July 10, 2023	Iterazione 4	Teresa Di Dona, Preziosa Pietroluongo	Project SAD
Raffinamento Grafici documentazione	In Pro...	July 13, 2023	Iterazione 4	Teresa Di Dona, Preziosa Pietroluongo	Project SAD
Caricamento server unina	In Pro...	July 11, 2023	Iterazione 4	Emanuele Messuri, Teresa Di Dona, Preziosa Pietr	Project SAD
Testing generale	In Pro...	July 11, 2023	Iterazione 4	Emanuele Messuri, Preziosa Pietroluongo, Teresa	Project SAD
Analisi EvoSuite	Done	April 14, 2023	Iterazione 1	Pasquale, Emanuele Messuri	Project SAD
Studio Integrazione	Done	April 14, 2023	Iterazione 1	Teresa Di Dona, Preziosa Pietroluongo	Project SAD
Creazione Diagrammi	Done	April 21, 2023	Iterazione 1	Teresa Di Dona, Preziosa Pietroluongo	Project SAD
Creazione prototipo 1.0	Done	April 21, 2023	Iterazione 1	Emanuele Messuri, Pasquale	Project SAD
Scrittura documentazione prima iterazione	Done	April 21, 2023	Iterazione 1	Emanuele Messuri, Teresa Di Dona, Preziosa Pietr	Project SAD
Creazione prototipo 2.0	Done	May 6, 2023	Iterazione 2	Emanuele Messuri, Pasquale	Project SAD
Approfondimento Metriche EvoSuite	Done	May 1, 2023	Iterazione 2	Teresa Di Dona, Preziosa Pietroluongo	Project SAD
Component Diagram	Done	May 6, 2023	Iterazione 2	Emanuele Messuri, Teresa Di Dona, Preziosa Pietr	Project SAD
Testing Prototipo con classi più complesse	Done	May 6, 2023	Iterazione 2	Emanuele Messuri, Pasquale	Project SAD
Scrittura documentazione II Iterazione	Done	May 6, 2023	Iterazione 2	Teresa Di Dona, Preziosa Pietroluongo	Project SAD
Approfondimento Docker	Done	May 16, 2023	Iterazione 3	Emanuele Messuri, Teresa Di Dona, Preziosa Pietr	Project SAD
Approfondimento API	Done	May 16, 2023	Iterazione 3	Emanuele Messuri, Teresa Di Dona, Preziosa Pietr	Project SAD
Approfondimento Node JS	Done	May 20, 2023	Iterazione 3	Emanuele Messuri, Teresa Di Dona, Preziosa Pietr	Project SAD
Creazione Prototipo Node JS	Done	May 28, 2023	Iterazione 3	Teresa Di Dona, Preziosa Pietroluongo	Project SAD
Adattamento Prototipo misurazioni	Done	May 28, 2023	Iterazione 3	Emanuele Messuri, Pasquale	Project SAD
Generalizzazione Prototipo generazione	Done	June 6, 2023	Iterazione 3	Emanuele Messuri, Pasquale	Project SAD
Configurazione container misurazione	Done	June 6, 2023	Iterazione 3	Teresa Di Dona, Preziosa Pietroluongo	Project SAD
Creazione Robot_generazione iterazioni	Done	June 30, 2023	Iterazione 4	Pasquale, Emanuele Messuri	Project SAD
Creazione Robot_generazione livelli	Done	July 7, 2023	Iterazione 4	Pasquale, Emanuele Messuri	Project SAD

Figura 3: Tasks

2. Storie Utente

Le **storie utente** hanno lo scopo di identificare e documentare in modo dettagliato le esigenze e le aspettative degli utenti finali. Esse descrivono le attività specifiche che gli utenti desiderano svolgere all'interno del sistema e offrono una guida per la progettazione e lo sviluppo delle funzionalità. Nelle storie utente di seguito mostrate, si farà riferimento al GamePlayer e all'amministratore che saranno gli utilizzatori del nostro sistema.

USER STORY TITLE:

MISURATION

AS I *GAMEPLAYER*

I WANT TO MAKE A REQUEST FOR THE MISURE COVERAGE AT ANY TIME

SO THAT I CAN KNOW THE EFFECTIVENESS OF MY TESTS

USER STORY TITLE:

SAVING FILE

AS I *GAMEPLAYER*

I WANT TO SPECIFY THE SAVING PATH

SO THAT I CAN CHOOSE THE DESTINATION OF THE FILE

USER STORY TITLE:

TEST GENERATION

AS I ADMINISTRATOR

I WANT TO START BUILDING THE TEST SUITES

SO THAT IS POSSIBLE TO CREATE THE LEVELS OF THE GAME

USER STORY TITLE:

LEVEL GENERATION

AS I ADMINISTRATOR

I WANT TO SPECIFY THE NUMBER OF LEVELS OF THE GAME

SO THAT I CAN CHOOSE HOW MANY DIFFICULTY LEVELS THE GAME SHOULD HAVE

2.1 Criteri di accettazione

Un criterio di accettazione è una descrizione dettagliata delle condizioni che devono essere soddisfatte affinché una user story sia considerata completata e accettata.

Questi criteri definiscono le aspettative precise dell'utente o del cliente rispetto alla funzionalità o al requisito specifico.

USER STORY TITLE:

MISURATION

AS I *GAMEPLAYER*

I WANT TO MAKE A REQUEST FOR THE MISURE COVERAGE AT ANY TIME

SO THAT I CAN KNOW THE EFFECTIVENESS OF MY TESTS

ACCEPTANCE CRITERION 1:

GIVEN A WRONG CLASS PATH

WHEN THE *GAMEPLAYER* SEND THE REQUEST

THEN THE SYSTEM RETURNS AN ERROR MESSAGE

AND THE MEASUREMENT CANNOT BE PERFORMED

ACCEPTANCE CRITERION 2:

GIVEN A WRONG TEST PATH

WHEN THE *GAMEPLAYER* SEND THE REQUEST

THEN THE SYSTEM RETURNS AN ERROR MESSAGE

AND THE MEASUREMENT CANNOT BE PERFORMED

ACCEPTANCE CRITERION 3:

GIVEN A RIGHT CLASS PATH

AND A RIGHT TEST PATH

WHEN THE *GAMEPLAYER* SEND THE REQUEST

THEN THE MEASUREMENT WAS PERFORMED SUCCESSFULLY

AND THE RESULTS ARE SAVED

USER STORY TITLE:

SAVING FILE

AS I *GAMEPLAYER*

I WANT TO SPECIFY THE SAVING PATH

SO THAT I CAN CHOOSE THE DESTINATION OF THE FILE

ACCEPTANCE CRITERION 1:

GIVEN A NON-EXISTENT FILE SAVE PATH

WHEN THE *GAMEPLAYER* SPECIFY THE SAVING PATH

THEN THE SYSTEM RETURNS AN ERROR MESSAGE

AND FILE SAVING CANNOT OCCUR

USER STORY TITLE:

TEST GENERATION

AS I ADMINISTRATOR

I WANT TO START BUILDING THE TEST SUITES

SO THAT IS POSSIBLE TO CREATE THE LEVELS OF THE GAME

ACCEPTANCE CRITERION 1:

GIVEN A WRONG CLASS PATH

WHEN THE *ADMINISTRATOR* START BUILDING THE TEST SUITES

THEN THE SYSTEM RETURNS AN ERROR MESSAGE

AND THE GENERATION DOESN'T HAPPEN

USER STORY TITLE:

LEVEL GENERATION

AS I ADMINISTRATOR

I WANT TO SPECIFY THE NUMBER OF LEVELS OF THE GAME

SO THAT I CAN CHOOSE HOW MANY DIFFICULTY LEVELS THE GAME SHOULD HAVE

ACCEPTANCE CRITERION 1:

GIVEN A NUMBER OF LEVELS IS EQUAL TO 0

WHEN THE ADMINISTRATOR SPECIFY THE NUMBER OF LEVELS OF THE GAME

THEN THE SYSTEM RETURNS AN ERROR MESSAGE

AND LEVELS CANNOT BE GENERATED

ACCEPTANCE CRITERION 2:

GIVEN A NUMBER OF PERMITTED LEVELS

WHEN THE ADMINISTRATOR SPECIFY THE NUMBER OF LEVELS OF THE GAME

THEN THE SYSTEM SUCCESSFULLY CREATE LEVELS

AND SAVE THEM

3. Documento dei requisiti

Il **Documento dei Requisiti** definisce le specifiche e le esigenze per il sistema che sarà sviluppato. Questo sistema avrà il compito di offrire un servizio offline di generazione dei test, divisi per livelli, e uno online di misurazione della copertura di una suite di test creata dall'utente. Il sistema sarà implementato come parte di una piattaforma o di un'applicazione più ampia, che richiede un'adeguata gestione. Inoltre, esso ha lo scopo di fornire una visione chiara e dettagliata dei requisiti funzionali e non funzionali del sistema da noi creato. È destinato agli sviluppatori, ai progettisti e agli stakeholder coinvolti nel progetto, al fine di garantire una comprensione comune delle funzionalità richieste e delle aspettative nei confronti del sistema.

3.1 Requisiti informali

Il sistema deve sia gestire una richiesta di un utente esterno, offrendo la possibilità di compilare ed eseguire casi di test per una determinata classe, che generare i diversi livelli del gioco. In entrambi i casi, il sistema deve consentire la misura della copertura del codice.

Per soddisfare questi requisiti, il sistema dovrà essere diviso in due parti:

- La prima gestisce la generazione dei livelli e comprende, per ogni livello, la creazione dei rispettivi test e la relativa misura della copertura. Tale misura e i relativi test, dovranno essere salvati in un repository condiviso.
- La seconda deve gestire la richiesta dell'utente effettuando una misurazione sui test forniti e salvando i risultati nel repository condiviso. Questo servizio dovrà essere attivo per tutta la durata del gioco.

Il servizio deve offrire una documentazione per aiutare colui che utilizza il sistema ad avviare la procedura.

3.2 Requisiti funzionali

R01. : Il servizio deve garantire la corretta compilazione ed esecuzione della classe sotto test

R02. : Il servizio deve restituire un messaggio di errore in caso di fallimento delle operazioni

R03. : Il servizio deve prevedere in ingresso le informazioni necessarie all'avvio dell'esecuzione

R04. : Il servizio deve garantire la corretta generazione dei test per i diversi livelli

R05. : Il servizio deve selezionare il numero di livelli richiesto

R06. : Il servizio deve fornire la misurazione della copertura per ogni livello richiesto, al fine di valutare l'efficacia dei test

R07. : Il servizio deve garantire che la generazione dei livelli sia offline rispetto al gioco

R08. : Il servizio deve garantire il corretto salvataggio dei test dei livelli nel repository condiviso

R09. : Il servizio deve offrire la misura della copertura dei test su richiesta dell'utente

R10. : Il servizio deve notificare al chiamante la corretta esecuzione, specificando il percorso di salvataggio della misura effettuata (il percorso viene restituito per aggiornare l'utente di eventuali cambiamenti della destinazione di salvataggio rispetto a ciò che aveva specificato)

R11. : Il servizio deve garantire che la misura della copertura su richiesta esterna sia online rispetto al gioco

R12. : Il servizio deve garantire il corretto salvataggio dei risultati nel repository condiviso

3.3 Requisiti non funzionali

Di seguito sono riportati i requisiti non funzionali, descritti mediante scenari degli attributi di qualità. Nel dettaglio, un tale scenario è esplicitato attraverso 6 elementi:

- **Stimuls (Stimolo),** ovvero l'evento che sollecita il sistema;
- **Source of Stimuls (Sorgente dello Stimolo)**, la sorgente da cui proviene lo stimolo;
- **Artifact (Artefatto)**, la parte del sistema sottoposta allo stimolo. Chiaramente può essere l'intero sistema;
- **Environment (Contesto)**, il contesto in cui si verifica lo stimolo;
- **Response (Risposta)**, come il sistema risponde allo stimolo;
- **Response Misure (Misura della Risposta)**, permette di valutare se la risposta del sistema è soddisfacente

I requisiti che abbiamo deciso di soddisfare sono:

- **RNF-1-** *Disponibilità:* un software è considerato disponibile quando è pronto e accessibile per l'installazione, l'esecuzione e l'interazione da parte degli utenti finali.

Parte dello Scenario	**Descrizione**	**Possibili valori**
Source	Specifica da dove viene il fallimento	Game Player
Stimulus	Possibili fault causati da richieste errate	Richiesta errata
Artifact	La porzione del sistema che subisce e gestisce l'errore	Server che gestisce la richiesta
Environment	Situazione del sistema quando arriva lo stimulus	Normale operatività
Response	Possibile reazione allo stimulus	Il sistema diventa momentaneamente non disponibile finché non viene riparato
Response measure	Percentuale di disponibilità desiderata e tempo di riparazione richiesto per ottenerla	Disponibilità: 99,9% Tempo di riparazione: 2 minuti

- **RNF-2-***Modificabilità:* Il sistema deve garantire la possibilità di modificare i suoi componenti e metodi facilmente.

Parte dello Scenario	Descrizione	Possibili valori
Source	L'agente che effettua la modifica	Developer
Stimulus	Il cambiamento che si vuole effettuare	Aggiungere/modificare le funzionalità, modificare gli attributi di qualità o la tecnologia
Artifact	Le componenti del sistema che vengono modificate	Il codice, le interfacce e la documentazione
Environment	Stato in cui il sistema può essere modificato	Sistema non operativo
Response	Come il sistema reagisce alle modifiche	Incorporare le modifiche e testarle
Response measure	Risorse necessarie per effettuare le modifiche	Meno di un giorno di lavoro

- **RNF-3-***Performance:* si riferisce alla capacità di un software o di un sistema di svolgere le proprie funzioni in modo efficiente ed efficace, rispettando determinate prestazioni e tempi di risposta definiti.

Parte dello Scenario	Descrizione	Possibili valori
Source	Lo stimolo può provenire da uno o più utenti	Game Player
Stimulus	Lo stimolo è l'arrivo di un evento, ovvero l'arrivo di una richiesta	Richiesta sporadica
Artifact	L'artefatto rappresenta l'intero sistema o la porzione di sistema che deve gestire la richiesta	L'intero sistema
Environment	Lo stato del sistema quando arriva lo stimolo	Operatività normale
Response	Come il sistema reagisce alla richiesta	Il sistema genera una misura e restituisce una risposta
Response measure	Tempistiche necessarie per servire la richiesta	Massimo 5 minuti

4. Tecnologie Utilizzate

Per lo sviluppo del progetto si è optato per l'utilizzo delle seguenti tecnologie:

- **Evosuite**

Figura 4: Evosuite Logo

Evosuite è un framework automatico di generazione di test per il software. L'obiettivo principale di Evosuite è quello di generare test che coprano diversi percorsi di esecuzione nel codice sorgente dell'applicazione, aiutando così gli sviluppatori a individuare bug, migliorare la qualità del software e facilitare il processo di debugging. Evosuite utilizza tecniche di ricerca evolutiva, come l'algoritmo genetico, per migliorare progressivamente la qualità dei test generati, cercando di massimizzare la copertura del codice e identificando possibili scenari di errore.

- **Node.js**

Figura 5: Nodejs Logo

Node.js è un ambiente di runtime open-source basato sul motore JavaScript V8 di Google. È progettato per consentire l'esecuzione di codice JavaScript lato server, consentendo agli sviluppatori di creare applicazioni web e servizi di rete ad alte prestazioni.

- **Maven**

Figura 6: Maven Logo

Maven è uno strumento per gestire le dipendenze e automatizzare la creazione di progetti software. Utilizzando Maven, è possibile definire il progetto tramite un file di configurazione denominato "pom.xml", specificando le dipendenze

esterne necessarie per il progetto.
Maven scaricherà automaticamente le dipendenze richieste e le includerà nel progetto. Inoltre, esso semplifica la compilazione, il testing e il packaging del progetto. In generale, Maven semplifica notevolmente il processo di gestione e compilazione dei progetti software, fornendo una struttura coerente e prevedibile per lo sviluppo.

- **Hamcrest**

Figura 7: Hamcrest Logo

Hamcrest è una libreria di asserzioni (assertion library) per il linguaggio di programmazione Java. Essa fornisce una sintassi espressiva per scrivere asserzioni più leggibili e comprensibili nei test unitari.

- **JUnit**

Figura 8: JUnit Logo

JUnit è un framework di test unitari per il linguaggio di programmazione Java. Esso fornisce una struttura e un insieme di API per scrivere, eseguire e verificare test unitari in modo automatizzato.

- **JDK 8**

Figura 9: JDK-8 Logo

JDK-8 è una versione del kit di sviluppo Java che include il compilatore, le librerie e gli strumenti necessari per creare, compilare ed eseguire applicazioni Java.

- **Docker**

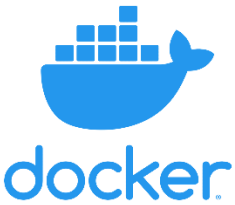


Figura 10: Docker Logo

Docker è una piattaforma per la creazione, la distribuzione e l'esecuzione di applicazioni in contenitori. Un contenitore è un'unità isolata che racchiude il codice, le librerie e tutte le dipendenze necessarie per eseguire un'applicazione.

- **Postman**

Figura 11: Postman Logo

Postman è un ambiente di sviluppo API completo. Consente agli sviluppatori di effettuare richieste API, ispezionare i dati di richiesta e risposta ed eseguire il debug degli endpoint API. Permette di salvare e organizzare richieste API, rendendo facile tenere traccia di diversi endpoint API e passare rapidamente tra loro, oltre a condividerle con altri, rendendolo uno strumento molto utile per la collaborazione.

5. Glossario

CFG: control flow graph

CUT: classe sotto test

LINE: per essere soddisfatto, questo criterio prevede che una test suite esegua ogni linea di codice non commentata almeno una volta.

BRANCH: una suite di test soddisfa questo criterio se tutti i rami del CFG sono valutati da almeno un caso di test: la suite di test, quindi, contiene un test per ogni nodo istruzione e, per i nodi predicato, almeno un test la cui esecuzione valuta il predicato del ramo true e, almeno uno, la cui esecuzione valuta il predicato del ramo false.

EXCEPTION: questo criterio premia le suite di test che costringono la CUT a generare più eccezioni possibili, dichiarate o non dichiarate.

WEAK MUTATION: una suite di test soddisfa questo criterio se, per ogni mutazione generata, almeno un test rileva la mutazione. Il test di mutazione debole apporta piccole modifiche al codice della classe sotto test, per controllare, se esiste un test in grado di distinguere tra l'originale e il mutante.

OUTPUT: in questo criterio i tipi degli output dei vari metodi, vengono mappati come valori astratti e una suite di test soddisfa il criterio se, per ogni metodo pubblico, almeno un test produce un valore concreto caratterizzato da ogni valore astratto.

METHOD: questo criterio richiede semplicemente che tutti i metodi della CUT vengano eseguiti almeno una volta, direttamente o indirettamente.

METHOD NO EXCEPTION: questo criterio richiede che tutti i metodi della CUT vengano chiamati in maniera diretta e che la loro esecuzione debba terminare senza generare eccezioni.

C-BRANCH (Context Branch): questo criterio forza la generazione di casi di test che richiamano in maniera diretta il metodo che contiene un ramo del CFG. Quindi, un ramo del grafo non viene considerato coperto se viene invocato da altri metodi.

6. Diagramma dei Casi d'uso

Il **diagramma dei casi d'uso** rappresenta le interazioni tra gli utenti e il sistema, mettendo in evidenza le funzionalità e le azioni che possono essere eseguite dagli attori coinvolti.

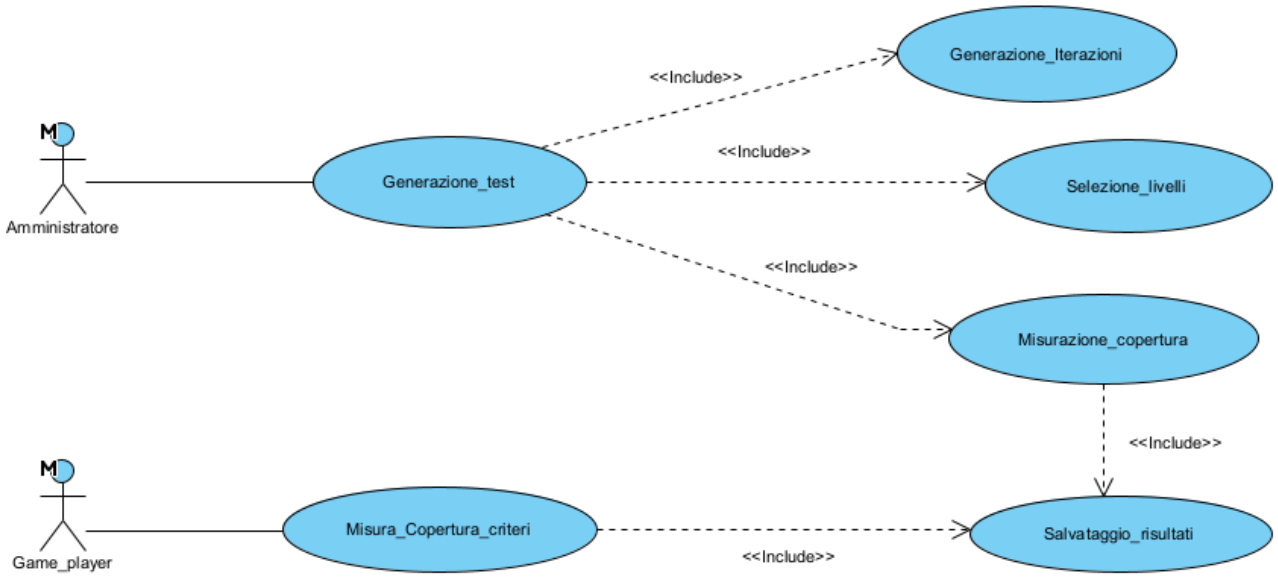


Figura 12: Diagramma dei casi d'uso

6.1 Scenari

Si riportano, di seguito, gli scenari d'uso per rappresentare in maniera più esplicita il comportamento del sistema:

CASO D'USO:	*GENERAZIONE TEST*
ATTORI:	Amministratore
DESCRIZIONE:	Generazione di una o più suite di test della classe sotto test
PRECONDIZIONI:	Disporre della classe da testare e specifica dei parametri di configurazione per la generazione dei test
SEQUENZA DEGLI EVENTI:	1. L'amministratore avvia la generazione dei test, indicando la classe da testare e il numero di livelli che si vuole generare. 2. Il sistema verifica che: • la classe sia presente • il numero di livelli sia valido 3. Il processo di generazione avviene attraverso Evosuite
POSTCONDIZIONE:	I vari test sono stati salvati nel repository condiviso
SEQUENZA ALTERNATIVA:	2.a La classe non è presente 2.b Il numero di livello non è valido oppure non è specificato
POSTCONDIZIONE	Il sistema segnala un messaggio di errore

CASO D'USO:	*GENERAZIONE_ITERAZIONI*
ATTORI:	Amministratore
DESCRIZIONE:	Generazione di diverse suite di test usando combinazioni diverse dei criteri di copertura
PRECONDIZIONI:	Disporre della classe da testare e specifica dei parametri di configurazione per la generazione dei test
SEQUENZA DEGLI EVENTI:	1. Viene avviata la generazione della suite di test che viene ripetuta su una combinazione diversa delle varie metriche di copertura.
POSTCONDIZIONE:	Le varie iterazioni sono state create e opportunamente salvate

CASO D'USO:	*SELEZIONE LIVELLI*
ATTORI:	Amministratore
DESCRIZIONE:	Selezione dei livelli tra le iterazioni generate
PRECONDIZIONE:	Disporre delle suite di test generate e del numero di livelli che si desidera generare
SEQUENZA DEGLI EVENTI:	1. Vengono selezionate le suite di test in base al numero di livelli
POSTCONDIZIONE:	I livelli prescelti sono stati selezionati

CASO D'USO:	*MISURAZIONE COPERTURA*
ATTORI:	Amministratore
DESCRIZIONE:	Misurazione della copertura delle suite di test selezionate
PRECONDIZIONI:	Disporre dei livelli selezionati e della classe sotto test
SEQUENZA DEGLI EVENTI:	1. Vengono effettuate le misurazioni della copertura delle suite di test selezionate in base a dei criteri prestabiliti
POSTCONDIZIONE	Le misurazioni sono state prodotte e le suite di test sono state salvate

CASO D'USO:	*MISURAZIONE COPERTURA TEST*
ATTORI:	Game_player
DESCRIZIONE:	Misurazione della copertura della suite di test fornita dal GamePlayer
PRECONDIZIONI:	Disporre della suite di test, della classe di test e che la suite di test sia valida
SEQUENZA DEGLI EVENTI:	1. Vengono effettuate le misurazioni della copertura della suite di test selezionate in base a dei criteri prestabiliti
POSTCONDIZIONE:	Le misurazioni sono state prodotte

CASO D'USO:	*SALVATAGGIO RISULTATI*
ATTORI:	Amministratore/Game_player
DESCRIZIONE:	Salvataggio dei risultati della misurazione della copertura delle suite di test
PRECONDIZIONI:	Disporre dei risultati della misurazione
SEQUENZA DEGLI EVENTI:	1. I risultati della copertura vengono salvati
POSTCONDIZIONE:	I risultati della misurazione sono disponibili nel repository condiviso per l'utilizzo

7. Diagramma delle classi

Nel diagramma delle classi è stato rappresentato concettualmente come sono collegati tra loro i componenti del nostro sistema.

Nel diagramma sottostante si evince la presenza di cinque classi: *CUT* (la classe sotto test fornita in input), *suite_test_utente* (la suite di test fornita dall'utente attraverso il path, di cui si richiede la misurazione della copertura), *evosuite*, *suite_test_evosuite* (le suite di test prodotte da Evosuite) e *risultato* (relativi alla misurazione effettuata). L'associazione tra *CUT* e *suite_test_utente* è di tipo 1-0…1 per indicare che l'utente per ciascuna classe può fornire zero o una suite di test, questo perché il nostro sistema è in grado di offrire il servizio per una suite di test alla volta. L'associazione tra *CUT* e *suite_test_evosuite* è di tipo 1-0…* perché Evosuite può generare più suite di test associate alla stessa classe. L'associazione di *suite_test_utente* e *suite_test_evosuite* con *risultato* è di tipo 1-1 per indicare che ad ogni suite di test corrisponde un risultato.

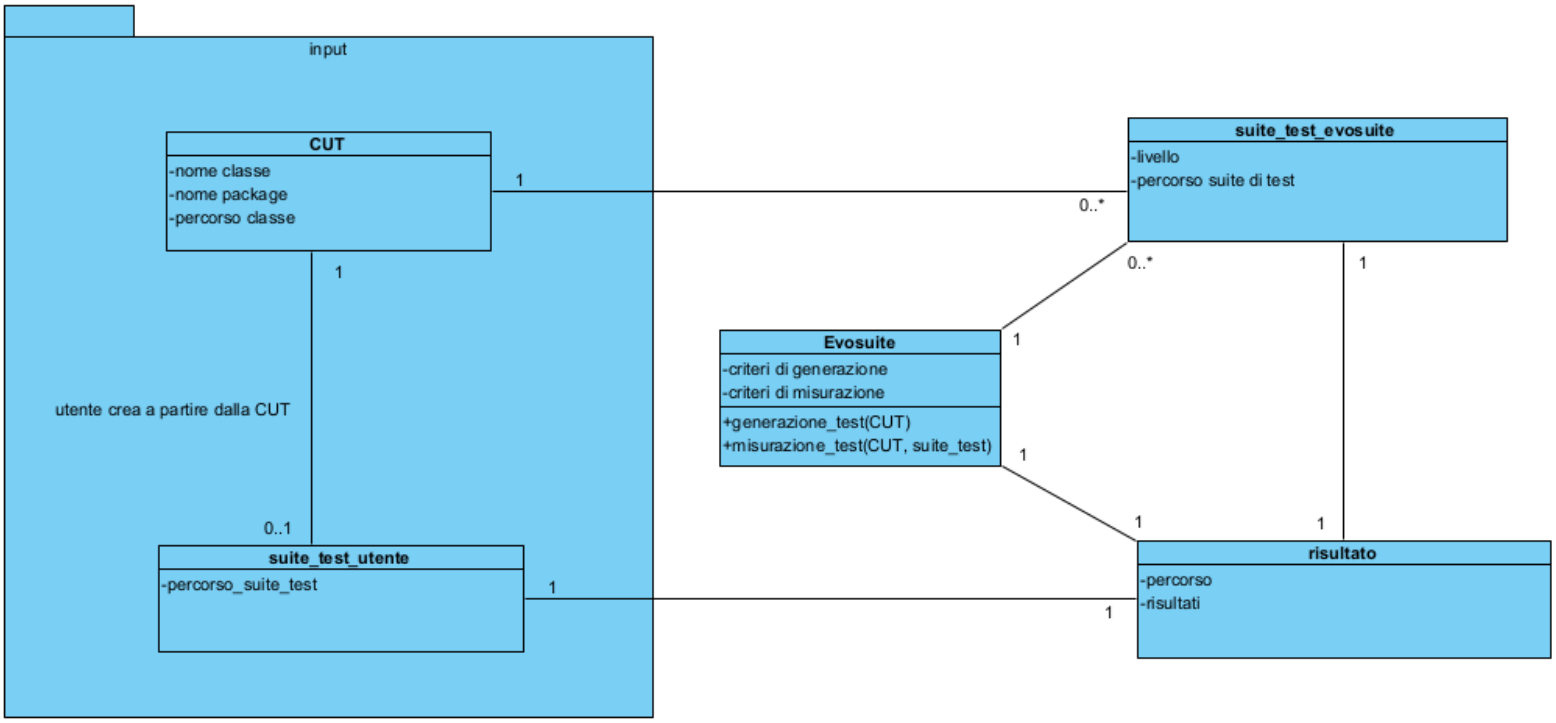


Figura 13: Diagramma delle classi

7.1 Risultati

Viene mostrato, di seguito, il formato del risultato prodotto. Il risultato è un file "*statistics.csv*" e nella seguente sezione si mostra la specifica dei suoi parametri attraverso un esempio sulla classe "*calendario*":

	A	B	C	D	E	F	G
1	TARGET_CLASS,criterion,Coverage,Total_Goals,Covered_Goals						
2	calendario.calendario,LINE,0.3333333333333333,48,16						
3	calendario.calendario,BRANCH,0.16091954022988506,87,14						
4	calendario.calendario,EXCEPTION,1.0,0,0						
5	calendario.calendario,WEAKMUTATION,0.2403846153846154,104,25						
6	calendario.calendario,OUTPUT,0.0,3,0						
7	calendario.calendario,METHOD,0.0,2,0						
8	calendario.calendario,METHODNOEXCEPTION,0.0,2,0						
9	calendario.calendario,CBRANCH,0.16091954022988506,87,14						
10							

Figura 14: Statistics.csv

- TARGET_CLASS: nome del package e nome della classe sotto test
- criterion: il criterio di valutazione dei test
- Total_Goals: numero totali di obiettivi da coprire
- Covered_Goals: numero di obiettivi coperti
- Coverage: Covered_Goals/ Total_Goal

8. Diagramma di contesto

Il **diagramma di contesto** rappresenta una panoramica ad alto livello del sistema in analisi e delle sue interazioni con gli attori esterni. Questo diagramma fornisce una visione chiara e concisa del modo in cui il sistema si integra con l'ambiente circostante e interagisce con gli altri componenti dell'applicazione.

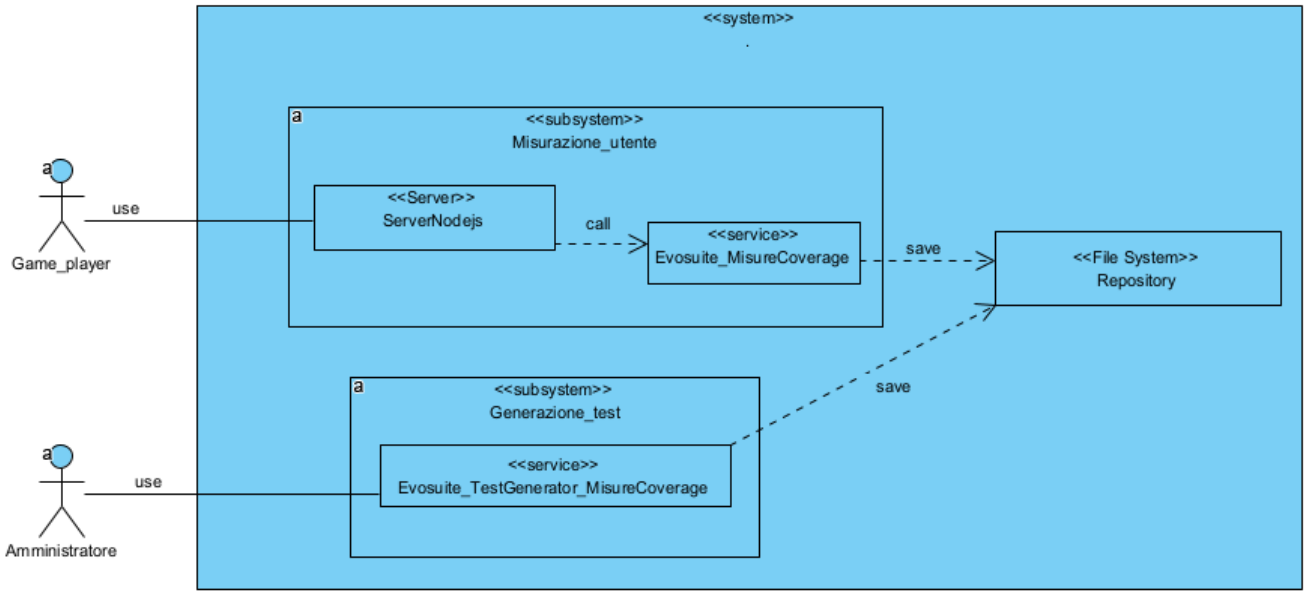


Figura 15: Diagramma di contesto

9. Misurazione utente

9.1 Scelte Progettuali

In questa sezione, riportiamo una descrizione della realizzazione del server con nodejs. Per la realizzazione del servizio, sono stati realizzati due file ".js": *prova_esecuzione_parametri4.js* e *child.js.*

Nel primo file, viene creato un server HTTP utilizzando la funzione *createServer* del modulo http. All'arrivo di una richiesta, viene controllato se l'URL inizia con '/api/'. In questo caso, la stringa dell'URL viene manipolata per ottenere i parametri necessari per l'avvio ed il corretto funzionamento della misurazione della copertura. Viene rimossa la parte *'/api/'*; viene estratto il primo elemento della richiesta, ovvero il percorso fino alla classe Java, e da quest'ultimo vengono estrapolati il path fino al package, il nome della classe Java e il nome del package. Viene, quindi, concatenata una stringa contenente tutti i parametri (nome della classe, nome del package, percorso del package, percorso dei test e percorso di salvataggio), la quale verrà inviata al file *robot_misurazione_utente.sh* che potrà iniziare con la sua esecuzione.

Per mantenere una coerenza temporale, abbiamo impostato il server in modo da lasciarlo in attesa durante l'esecuzione del bash che effettua la misurazione, per permettergli di terminare il proprio lavoro e di creare e salvare il file *statistics.csv*. In particolare, abbiamo creato un server figlio (ovvero il secondo file, child.js) con una fork, che attende una richiesta di terminazione da parte del bash. Una volta ricevuta, il Server "figlio" invierà un messaggio al "padre" che, nel mentre, era in attesa. A questo punto, l'operazione del server è completata e il client, in risposta, riceverà la conferma del successo dell'operazione.

9.2 Diagramma dei componenti

Il diagramma dei componenti viene utilizzato per descrivere l'architettura software di un sistema, esso mostra i componenti del sistema e le relazioni tra di essi.

Il seguente diagramma mostra i componenti utilizzati per effettuare la misura della copertura da parte degli utenti su richiesta esterna. L'utente effettua la richiesta al server Nodejs, che avvia l'esecuzione del componente Robot. Tale componente utilizza Evosuite per eseguire le operazioni necessarie alla misurazione. Evosuite usufruisce dei servizi offerti da Maven, Hamcrest-core e junit.

In particolare:

• Maven viene adoperato per la gestione delle dipendenze e la compilazione dei progetti Java

• Hamcrest-core viene impiegato per il supporto alla scrittura delle asserzioni, ovvero fornisce un insieme di costrutti che consentono di specificare in modo chiaro e dichiarativo le aspettative sui risultati dei test.

• JUnit viene utilizzato per scrivere ed eseguire unit testing per java e offre meccanismi per definire asserzioni e valutare i risultati dei test.

Tutti i componenti dipendono da "jdk", ovvero il kit di strumenti usati dal programmatore java. Per leggibilità nel diagramma, si è evitato di rappresentare tutti i collegamenti ad esso.

Terminate le operazioni di misurazione, viene salvato sul repository condiviso il file ".csv".

Si sottolinea come il robot, affinché possa essere avviata la misurazione, debba necessariamente prelevare, da tale repository, la classe e i test specificati dal GamePlayer.

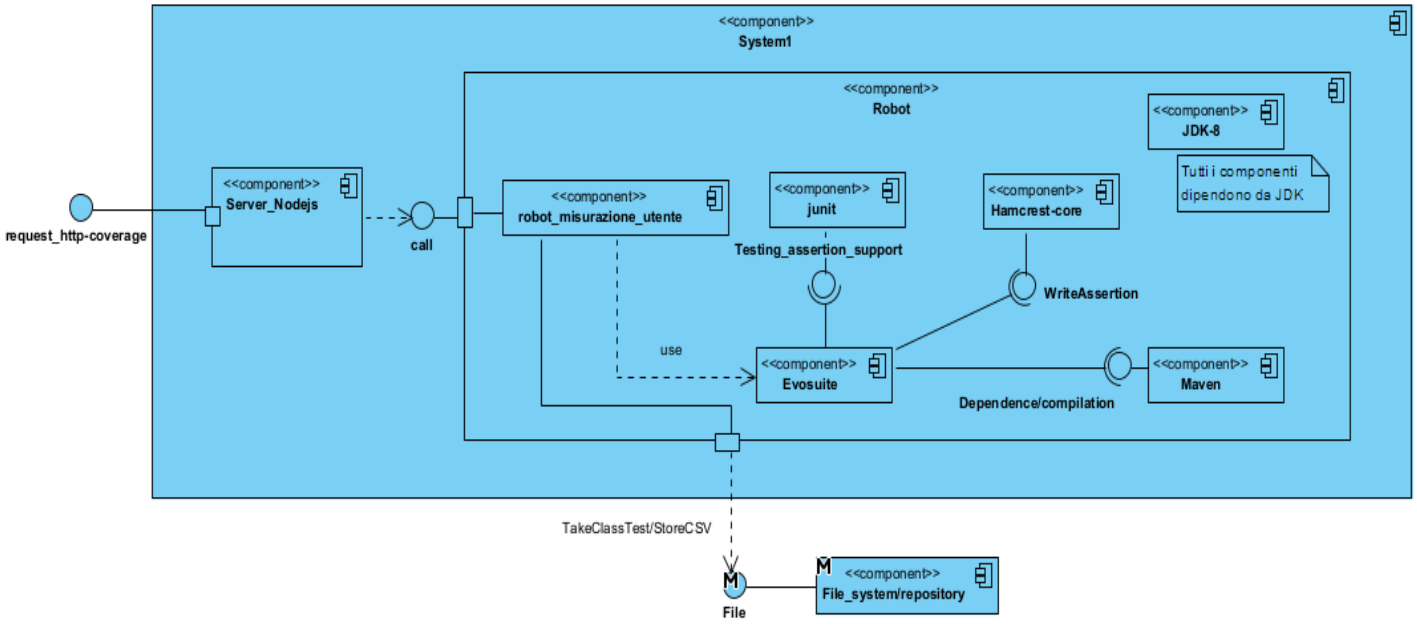


Figura 16: Diagramma dei componenti misurazione

9.3 Diagramma di attività

Nel diagramma di seguito mostrato, viene rappresento il flusso di esecuzione del nostro sistema. Si osservi come, tale flusso, ad un certo punto, venga scisso per distinguere il comportamento relativo al Robot, che si occupa della misurazione, e la creazione di un processo figlio che resterà in attesa della terminazione del file “bash”.

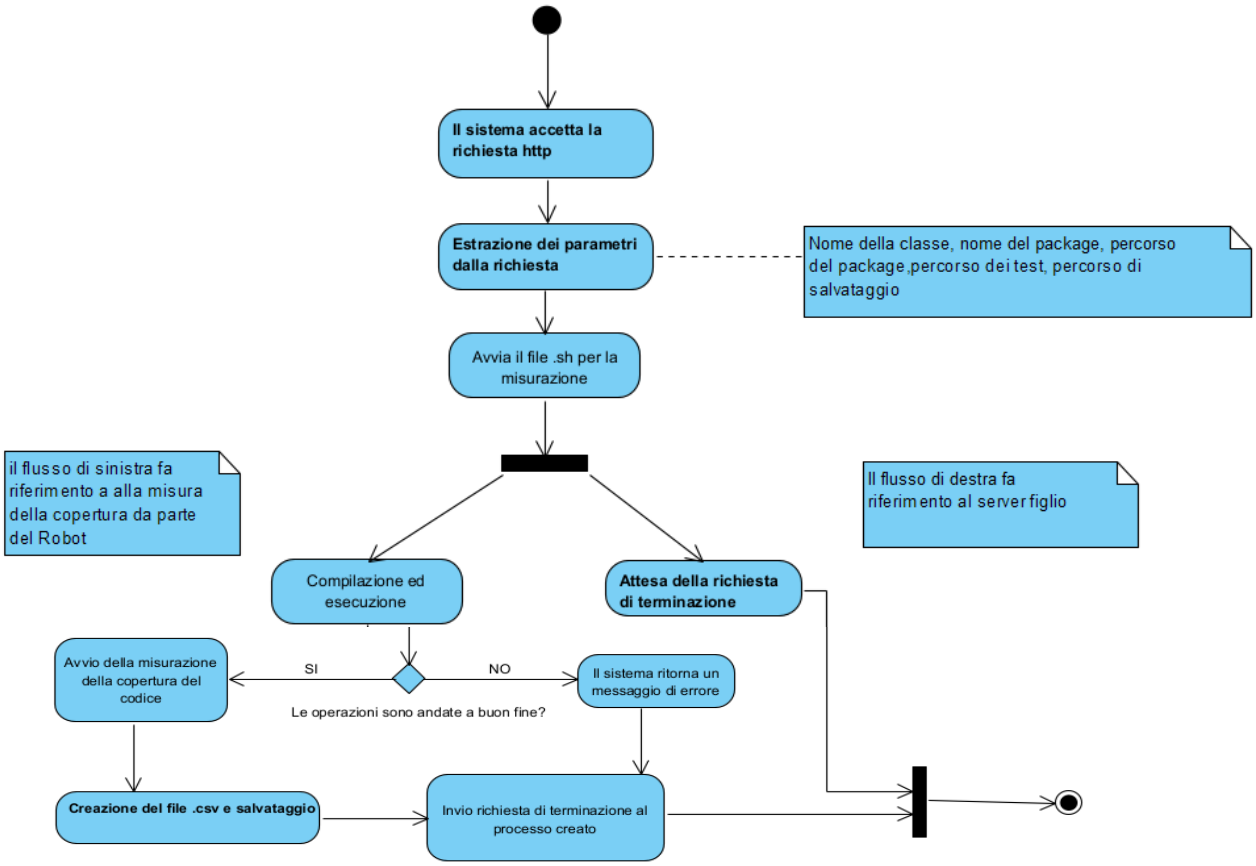


Figura 17: Diagramma di attività misurazione

9.4 Diagramma di comunicazione

Il **diagramma di comunicazione** è uno strumento visuale utilizzato per rappresentare le interazioni tra i diversi attori o componenti di un sistema. È particolarmente utile nella progettazione del software per modellare e comprendere le relazioni e le interazioni tra gli oggetti o le entità coinvolte nel sistema, consentendo agli sviluppatori di comprendere meglio la struttura del sistema e le dinamiche delle comunicazioni tra le diverse parti.

Dal diagramma sottostante, si può osservare come i componenti *ServerFather*, *ServerChild* e *Robot* siano componenti attivi. Il *Robot* alla fine della sua esecuzione produrrà il risultato della misurazione richiesta. Quando il *ServerFather* riceve la richiesta dal *Game_player*, esso da un lato avvia il processo del *Robot* e dall'altro lato crea un *serverChild*. Quest'ultimo è stato realizzato con l'obiettivo di sincronizzare la risposta (alla richiesta del *GamePlayer*) con l'effettiva produzione del risultato del *Robot*. Quindi, il *ServerFather* dopo aver avviato il *Robot,* creato il figlio, si pone in attesa del messaggio da quest'ultimo; il *ServerChild* rimane in attesa della richiesta di terminazione dopo la sua creazione (fork) e, una volta ricevuta, si ricongiunge al padre. Si noti come, il *ServerFather,* dopo aver avviato il processo e prima di procedere, debba ricevere un messaggio di corretto/errato avvio del *Robot*. Tale messaggio non rappresenta la fine del processo del *Robot.*

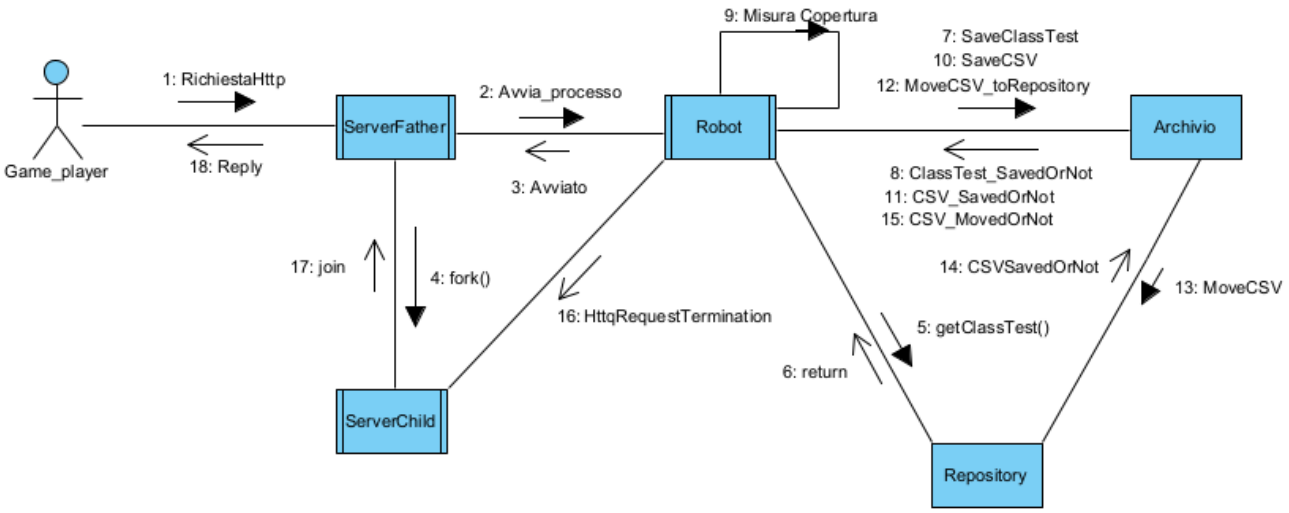


Figura 18: Diagramma di comunicazione Misurazione

9.5 Diagramma di sequenza

Il **Diagramma di sequenza** rappresenta la sequenza temporale delle interazioni tra gli oggetti principali all'interno del sistema. Questo diagramma offre una visione dettagliata dei passaggi e delle operazioni coinvolte nel processo di misurazione dei test.

In questo diagramma e come si potrà osservare anche in quelli illustrati successivamente, notiamo la creazione del file "pom.xml". Quest'ultimo è un file di configurazione fondamentale in un progetto Maven. Contiene informazioni come le dipendenze del progetto (librerie esterne), configurazioni di build e test.

Maven si occuperà di verificare se le dipendenze specificate all'interno del pom siano presenti o meno nel sistema e, se così non fosse, si occupa di scaricarle opportunamente.

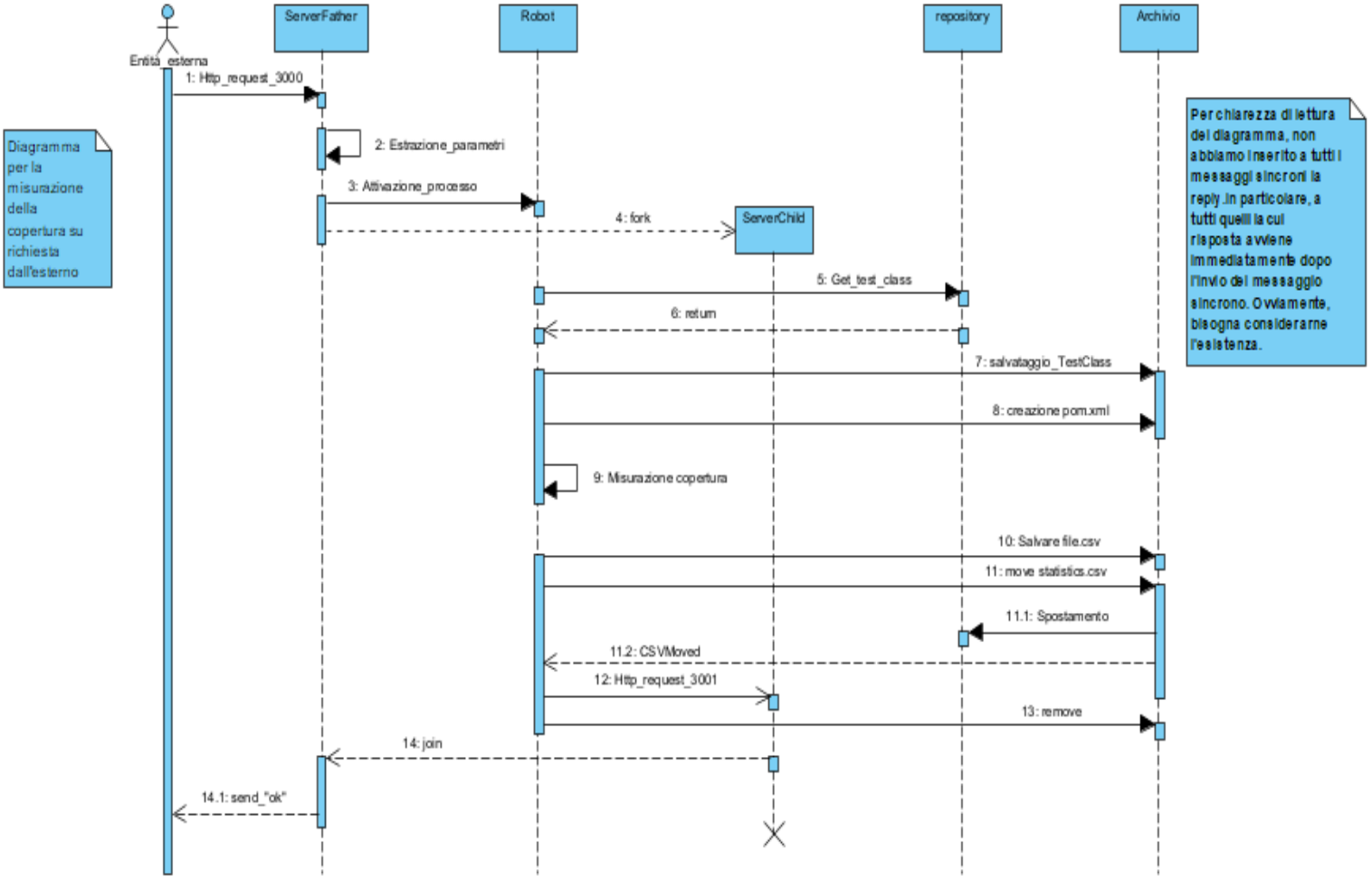


Figura 19: Diagramma di sequenza Misurazione

9.6 Formato della richiesta

Di seguito, viene riportato il formato della richiesta http:

http://ip:porta/api/percorso_classe+percorso_test+percorso_salvataggio

- ip:porta: IP e porta su cui è connesso e comunica il server. Il numero della porta scelta è stato 3000.
- Percorso_classe: Percorso della classe da testare; in questo percorso deve essere presente sia il nome della classe che il nome del package in cui essa è contenuta.
- Percorso_test: Percorso della suite di test
- Percorso_salvataggio: Percorso della cartella in cui salvare i risultati ottenuti; il nome del file, che verrà salvato, sarà “statistics.csv”.

Per visualizzare la Rotta API di tale richiesta si rimanda al capitolo *GUIDA ALL’UTILIZZO* nel paragrafo *ROTTA API*.

10. Generazione Test

10.1 Scelte progettuali

Durante la fase di progettazione della parte riguardante la generazione dei livelli, si è deciso di usare un algoritmo che campionasse, a partire da 13 iterazioni, le N classi di test necessarie alla creazione dei livelli richiesti in input. Per la generazione delle iterazioni, abbiamo scelto dei set di criteri di generazione, con lo scopo di ottenere valori di copertura diversi tra loro. I set scelti per le varie iterazioni sono:

1. Tutte
2. Line
3. Branch
4. Weak Mutation
5. Output
6. Method
7. Line-branch
8. Output-line
9. Weak Mutation-branch
10. Weak Mutation-output
11. C-branch-line
12. Line-branch-exception
13. Output-line-weak mutation

I set 2-6 hanno l'obiettivo di creare dei test che vadano a verificare le caratteristiche "base" della classe da testare, mentre i set successivi sono stati creati a partire da evidenze sperimentali ottenute eseguendo i set su varie classi diverse. In generale sono stati scelti dei set che producessero dei valori di copertura diversi tra loro.

10.2 Diagramma dei componenti

Come si evince dal diagramma, le relazioni tra i componenti sono analoghe a quelle viste per il diagramma della misurazione utente.

In questo diagramma, però, troviamo i tre componenti necessari alla generazione dei livelli, ovvero *Robot_generazione* (preleva la classe dal repository e crea le iterazioni), *generazione_livelli* (seleziona i livelli scelti) e *misurazione_livelli* (effettua le misurazioni della copertura e salva i risultati e le suite di test sul repository), che interagiscono tra loro.

Il componente *Robot_generazione* avvia la *generazione_livelli* e il componente *generazione_livelli* avvia *misurazione_livelli* e ne attende la corretta terminazione.

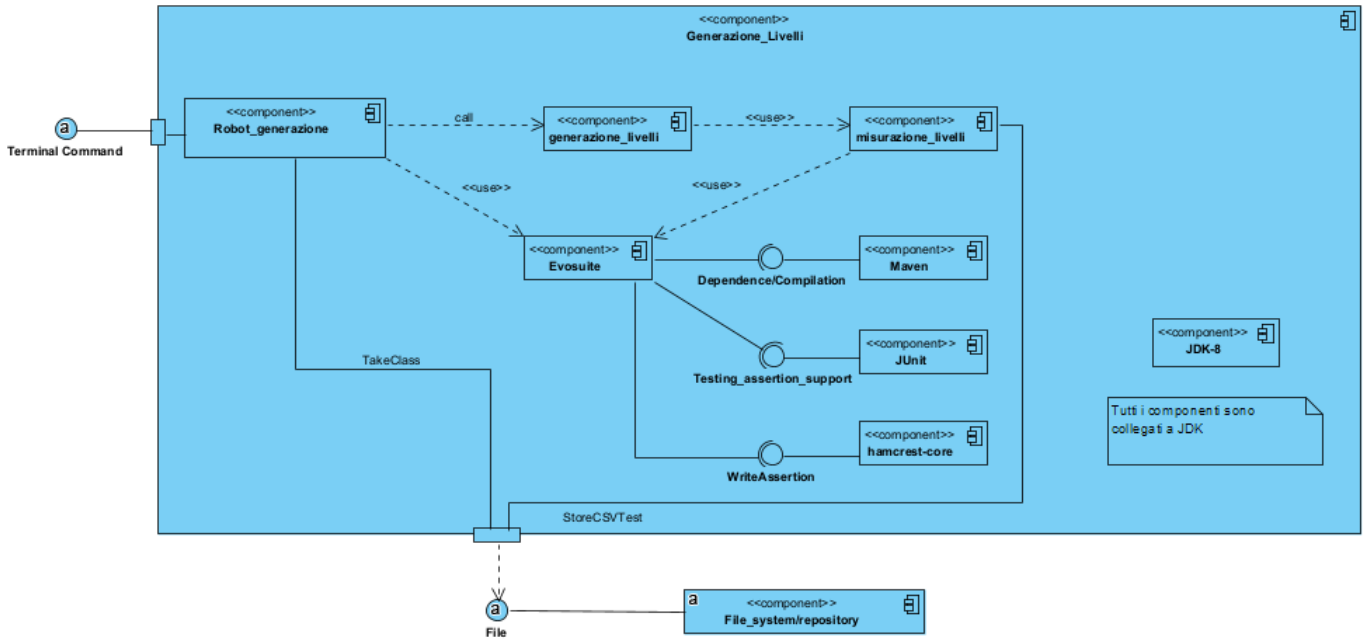


Figura 20: Diagramma dei componenti Generazione

10.3 Diagramma di attività

Il **diagramma di attività** offre una rappresentazione visuale del flusso di lavoro o dei processi all'interno del sistema. Questo diagramma consente di modellare le attività, le decisioni e le scelte durante il processo, fornendo una panoramica chiara e intuitiva delle azioni eseguite dagli utenti e dal sistema.
Il seguente grafico mostra il comportamento dei tre componenti:

- *Robot generazione*: componente che si occupa della generazione dei test

- *Generazione livelli*: presi in ingresso i test, crea un hash map con essi, la ordina e seleziona i livelli mediante l'*algoritmo di selezione livelli*.
- *Misurazione livelli*: prende in input i livelli selezionati, effettua le misurazioni di copertura e le salva nella relativa cartella del livello sul repository

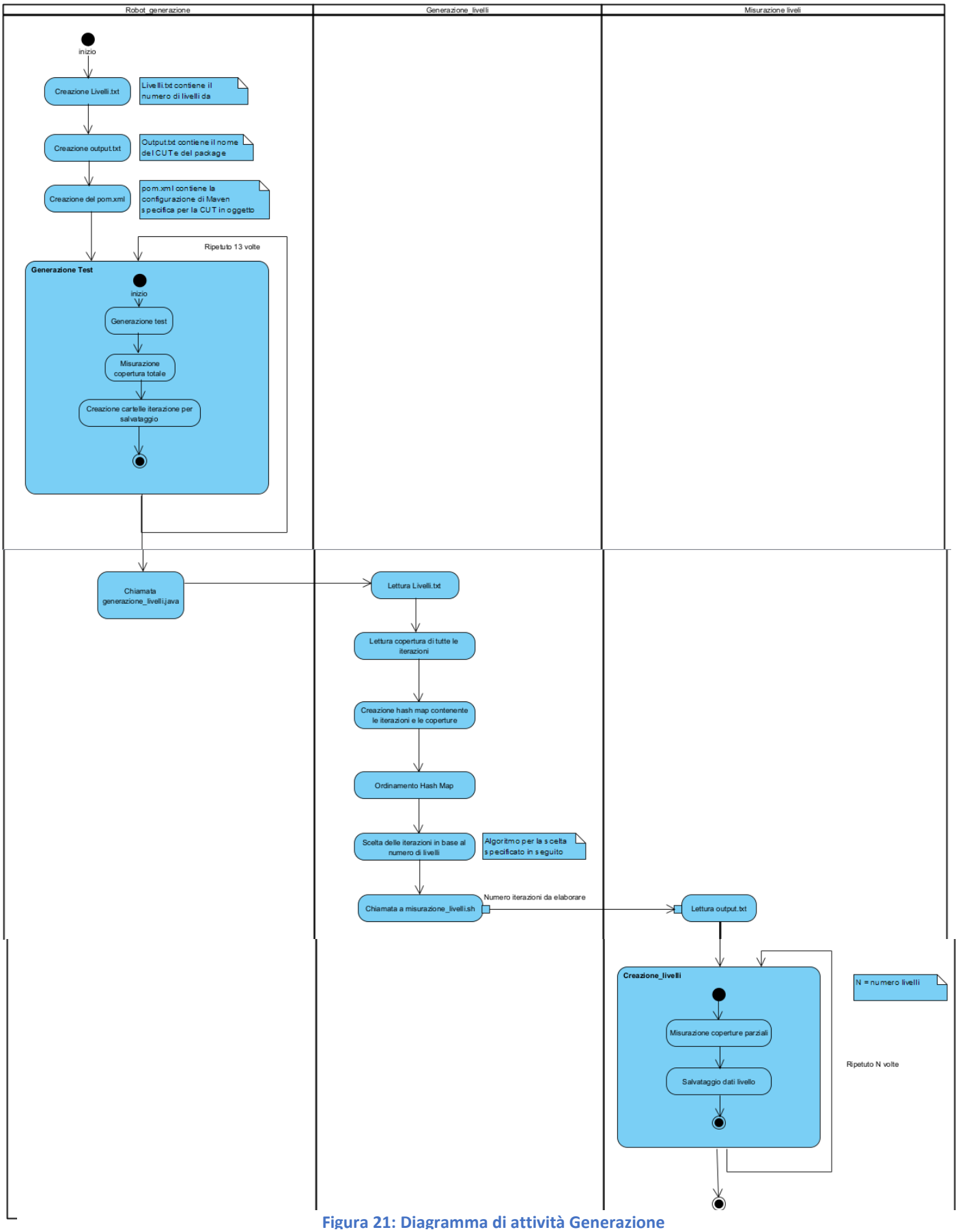


Figura 21: Diagramma di attività Generazione

10.4 Diagramma di sequenza

Come è stato già osservato nel diagramma delle attività, sono presenti anche nel diagramma di sequenze i seguenti loop:

• il loop per la generazione dei test e la corrispondente misurazione della copertura viene eseguito 13 volte per la generazione delle 13 iterazioni, a partire dalle quali verranno selezionati gli n livelli.

• i due cicli annidati si riferiscono alla misurazione della copertura per gli 8 criteri di ciascuno degli n livelli scelti. Per ciascun livello, le misurazioni ottenute vengono salvate nel repository in comune.

Così come riportato nel diagramma di sequenza per la misurazione_utente, anche in questo caso, per chiarezza di lettura, non abbiamo inserito a tutti i messaggi sincroni la reply. In particolare, a tutti quelli la cui risposta avviene immediatamente dopo il messaggio sincrono. Ovviamente, bisogna considerarne l'esistenza.

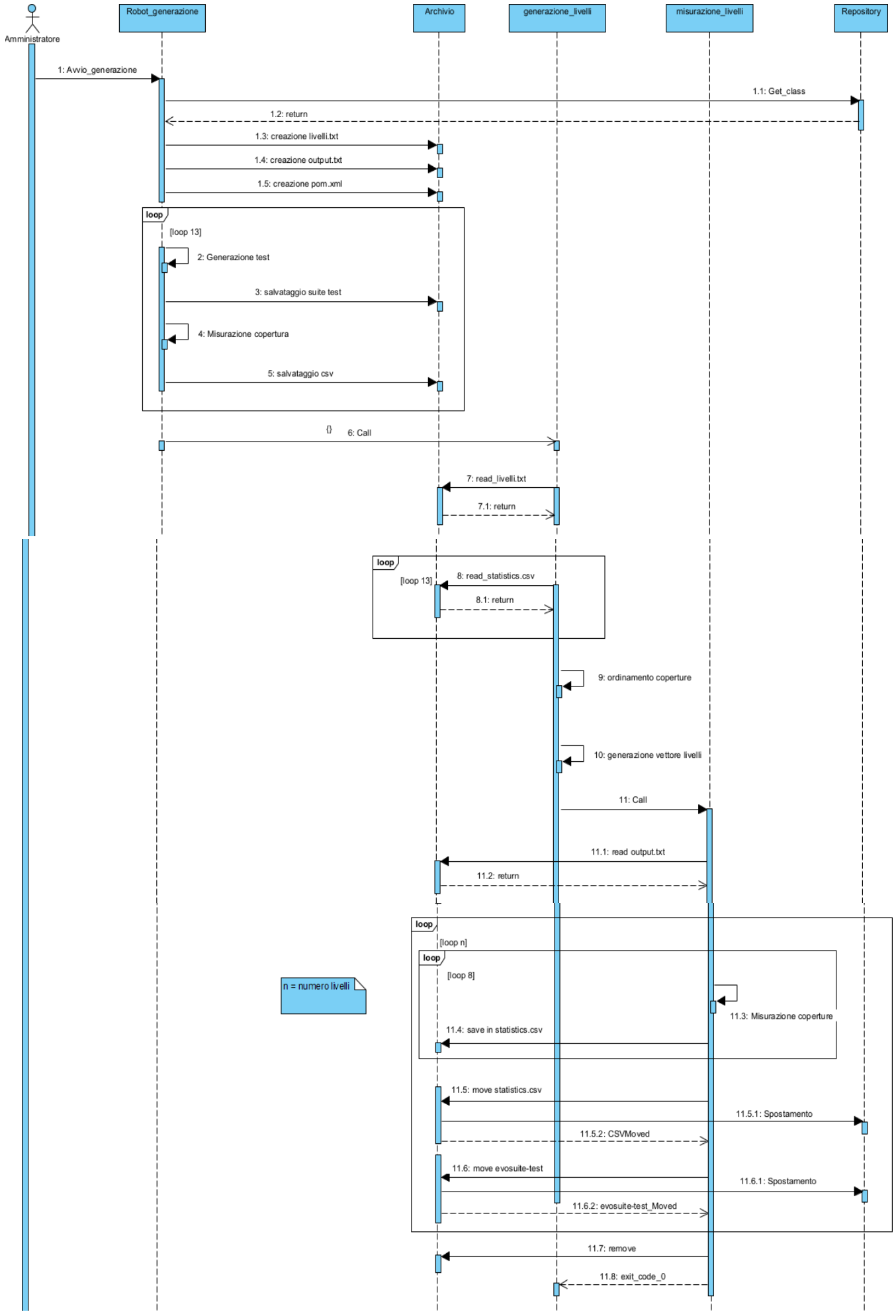


Figura 22: Diagramma di sequenza Generazione

10.5 Algoritmo di selezione dei livelli

Di seguito viene riportata una breve spiegazione dell'algoritmo di selezione dei livelli, per una maggiore chiarezza è stato creato un flowchart. Nell'algoritmo vengono usate come variabili:

- n: è l'input dell'algoritmo e indica il numero di livelli richiesto.
- spiaz: è lo spiazzamento che viene calcolato come 13 (numero massimo di livelli) diviso n, viene utilizzato per quantificare il passo di campionamento dell'algoritmo.
- r: indice del for.

L'algoritmo seleziona dal vettore ordinato, creato in precedenza, n livelli. Parte verificando il valore di n, se questo è maggiore di 1 allora aggiunge alla selezione l'elemento del vettore di posizione 1 e calcola lo spiazzamento, in caso negativo calcola direttamente lo spiazzamento. Dopo aver fatto questo, effettua una verifica sulla parità di n:

- se **n è pari**, tramite un ciclo for, selezioniamo $\frac{n-2}{2}$ elementi nella prima metà del vettore (il motivo di n-2 è che i primi due livelli selezionati saranno sempre l'elemento di posizione 1 e quello di posizione 12, di conseguenza l'algoritmo dovrà selezionare solo i restanti) e $\frac{n-2}{2}$ elementi nella seconda metà; la selezione avviene campionando il vettore secondo lo spiazzamento
- se **n è dispari** il funzionamento è simile con l'unica differenza che alla selezione viene aggiunto sempre anche il valore di posizione 7.

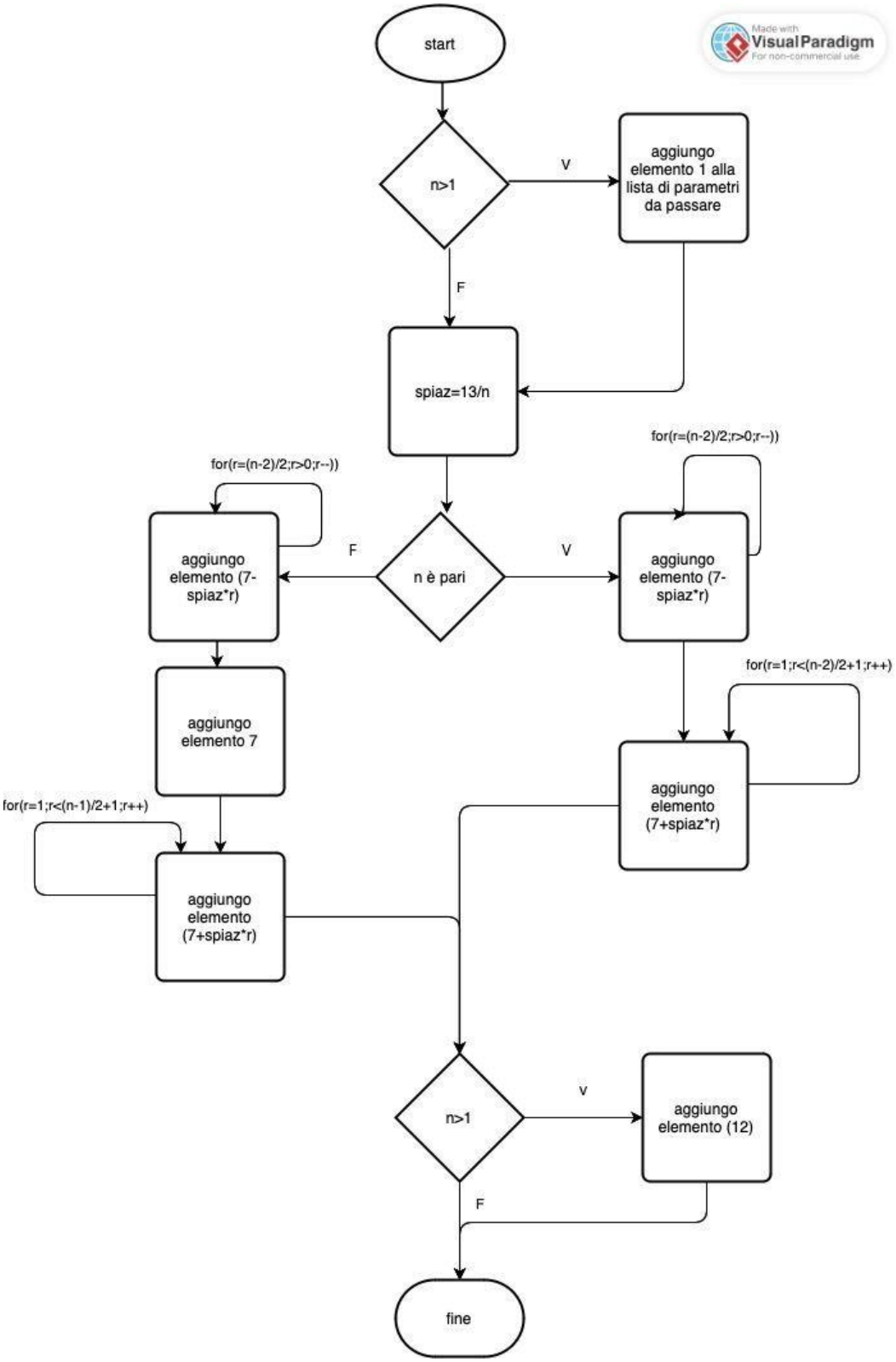


Figura 23: Flow Chart Algoritmo Generazione

10.6 Pipe and filter

"Pipe and Filter" è un pattern architetturale che prevede il flusso dei dati attraverso una serie di componenti chiamati "filtri" connessi tra loro da "pipes" (tubi). Ogni filtro è responsabile di elaborare i dati in input in base a una determinata funzione o trasformazione, e le pipes gestiscono il passaggio dei dati tra i filtri.

Questo stile architetturale è basato sul principio della separazione delle preoccupazioni (separation of concerns), in cui ogni filtro si occupa di un'attività specifica e indipendente, lavorando in modo isolato dagli altri filtri. I dati fluiscono attraverso le pipes da un filtro all'altro, consentendo una pipeline di elaborazione sequenziale.

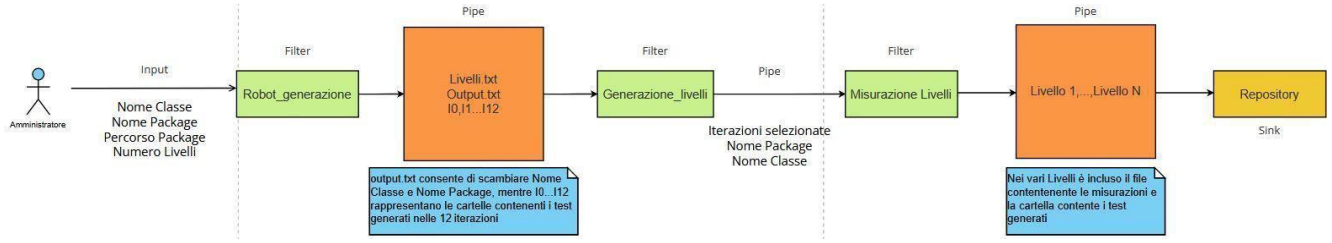


Figura 24: Pipe and Filter Generazione

Le caratteristiche principali dello stile Pipe and Filter includono:

1. Separazione dei compiti: Ogni filtro si concentra su una funzionalità specifica, come il filtraggio, la trasformazione, l'elaborazione, la validazione, ecc. Questo permette una migliore gestione della complessità e la possibilità di riutilizzare i filtri in contesti diversi.

2. Comunicazione tramite pipes: Le pipes connettono i filtri consentendo il flusso dei dati da un filtro all'altro. I filtri non sono a conoscenza degli altri filtri nella pipeline, comunicano solo attraverso le pipes.

3. Trasparenza dei dati: I filtri sono isolati l'uno dall'altro e non hanno conoscenza diretta dei dati che attraversano la pipeline. Trattano solo l'input ricevuto e producono un output, garantendo che la pipeline sia modulare e scalabile.

Questo stile architetturale è particolarmente utile quando si desidera separare le fasi di elaborazione così che possano essere gestite in modo indipendente e scalabile.

11. Deployment

11.1 Diagramma di Deployment

Il **diagramma di deployment** fornisce una visualizzazione della distribuzione fisica dei componenti del sistema, illustrando come le diverse parti del sistema sono organizzate e collocate su hardware e infrastrutture specifiche. Questi diagrammi consentono di comprendere l'ambiente di deploy del sistema e le relazioni tra i componenti hardware e software.
Per la realizzazione del nostro sistema, abbiamo utilizzato due container ubuntu docker:

- sul primo viene soddisfatta la richiesta dell'utente. Il sistema deve prevedere un accesso ad un repository condiviso sia per prelevare la classe sotto test e i test stessi, che per il salvataggio del file "statistics.csv". Questo container sarà attivo per tutta la durata del gioco.

- sul secondo viene gestita la generazione dei livelli che comprende, per ogni livello, la creazione dei rispettivi test e la relativa misura della copertura. Tale misura (salvata in un file "statistics.csv") e, i relativi test, dovranno essere salvati nell'opportuna directory di un repository condiviso. Questo container, una volta generati i test, verrà spento a tempo di gioco.

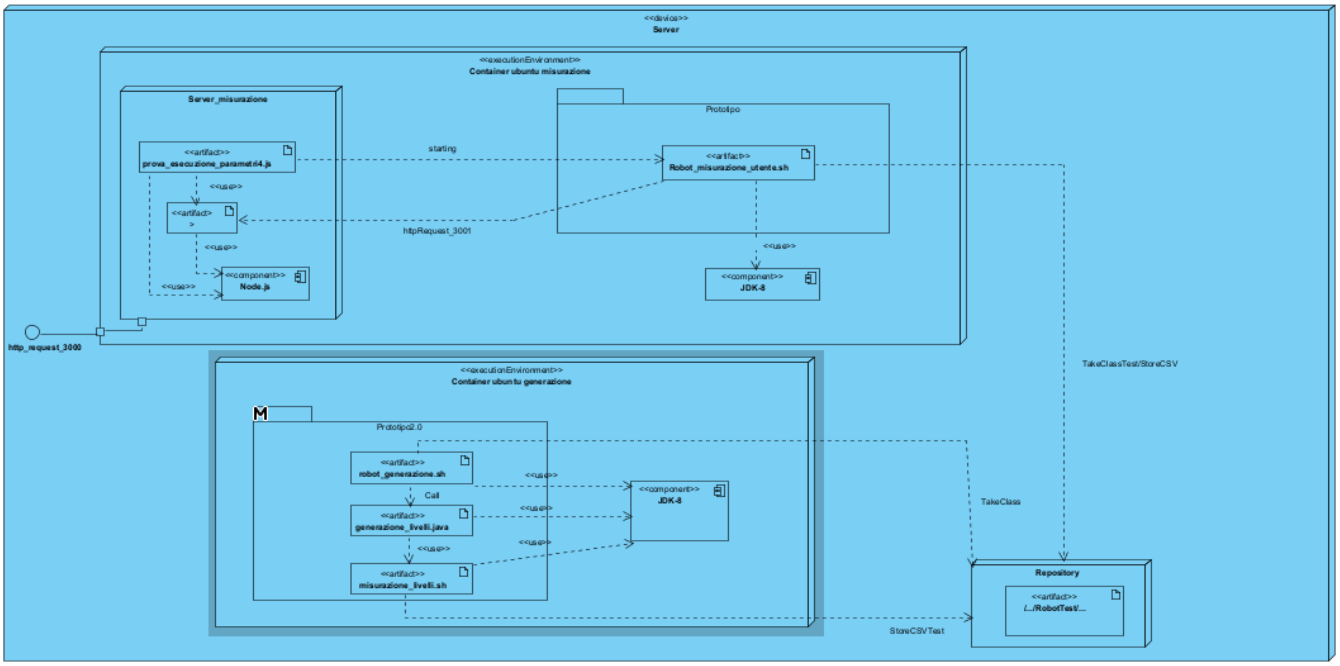


Figura 25: Diagramma Deployment

11.2 Installation view

L'**installation view** del sistema fornisce una rappresentazione visuale delle componenti e delle dipendenze necessarie per installare e configurare correttamente il sistema. Questo strumento è fondamentale per comprendere l'ambiente di installazione e i requisiti necessari per il corretto funzionamento del sistema.

Il file "installation.sh" si occupa di scaricare tutti i file ".jar" necessari per il funzionamento del nostro sistema.

Il file evosuite-1.0.6.jar è il nucleo di EvoSuite che gestisce la generazione dei casi di test, mentre il file evosuite-standalone-runtime1.0.6.jar è utilizzato per l'esecuzione dei test generati in un ambiente autonomo. Entrambi i file sono parte del pacchetto EvoSuite.

Il file hamcrest-core-1.3.jar è un file JAR contenente la libreria Hamcrest nella sua versione 1.3.

Il file Junit-4.13.1.jar è un file JAR contenente la libreria JUnit nella sua versione 4.13.1.

Per semplicità di visualizzazione, la cartella dependency, contenuta in /Prototipo2.0/target è stata riportata sopra nelle seguenti rappresentazioni dell'installation view.

11.2.1 Misurazione utente

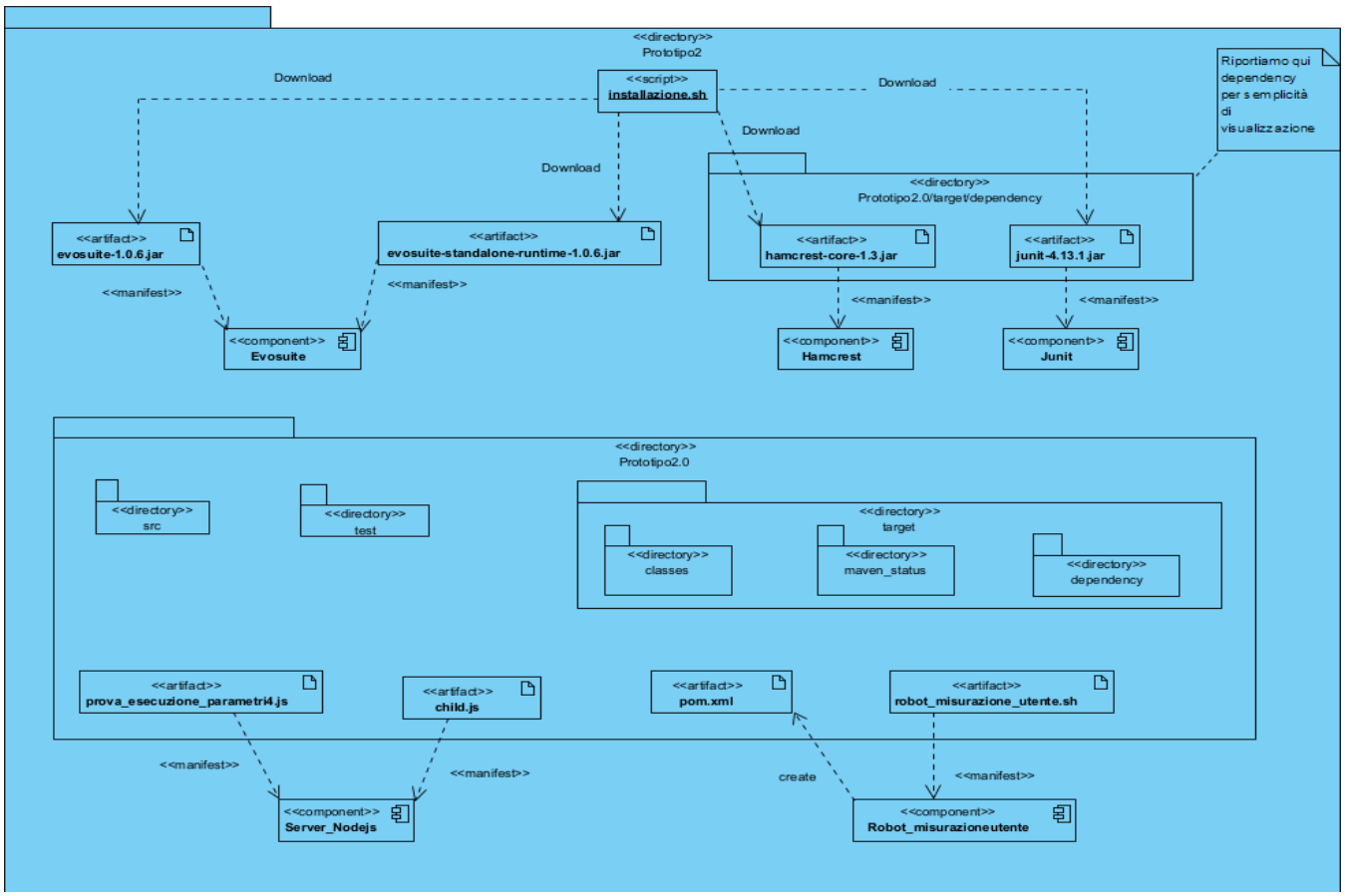


Figura 26: Diagramma di Installation View Misurazione

11.2.2 Generazione livelli

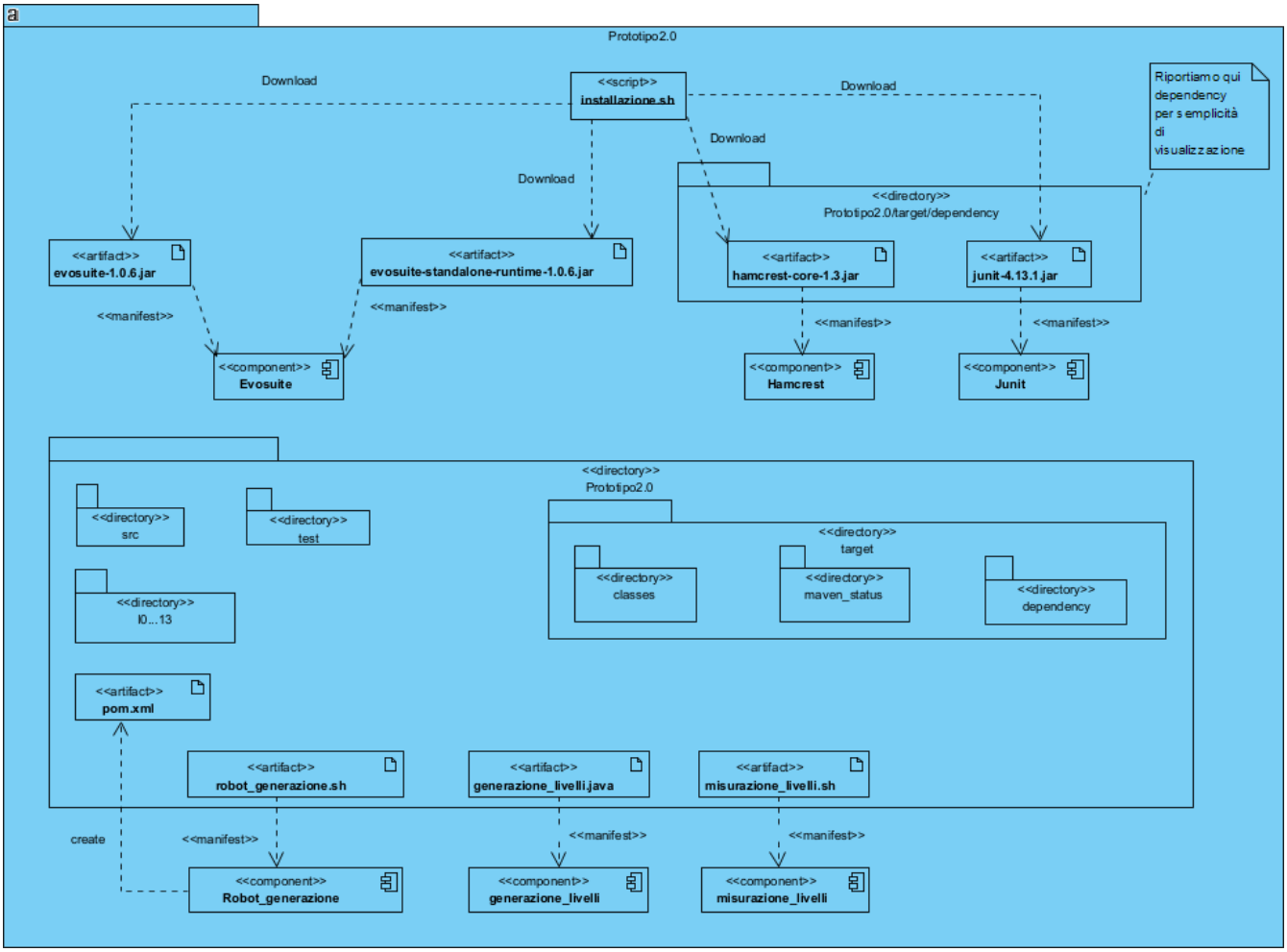


Figura 27: Diagramma di Installation view Generazione

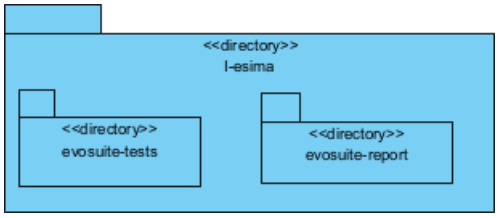


Figura 28: Directory Iterazione

12. Testing

12.1 Casi di test

Test ID	Descrizione	Pre-Condizioni	Input	Output	Post-Condizioni	Esito
1	Generazione Test 6 Livelli	La classe calcolatrice si trova nel percorso specificato nell'input	Calcolatrice, Calcolatrice, /percorso_package/ 6	Statistics.csv ed evosuite_test per ogni livello	Nelle cartelle dei primi 6 livelli sono stati inseriti gli output	PASS
2	Generazione Test 5 Livelli	La classe calendario si trova nel percorso specificato nell'input	Calendario, Calendario, /percorso_ package / 5	Statistics.csv ed evosuite_test per ogni livello	Nelle cartelle dei primi 5 livelli sono stati inseriti gli output	PASS
3	Generazione Test 3 Livelli con percorso sbagliato	La classe calendario non si trova nel percorso specificato nell'input	Calendario, Calendario, /percorso_package errato/ 3	Messaggio di errore	Nelle cartelle dei primi 3 livelli non sono stati inseriti gli output	PASS
4	Generazione Test 0 livelli	La classe calcolatrice si trova nel percorso specificato nell'input	Calcolatrice, Calcolatrice, /percorso_ package / 0	Messaggio di errore: "Arithmethic Exception: / by zero"	Nelle cartelle dei livelli non sono stati inseriti gli output	PASS

5	Generazione Test 3 Livelli	La classe ImprovedTokenizer si trova nel percorso specificato nell'input	ImprovedTokenizer, ImprovedTokenizer /percorso_ package / 3	Statistics.csv ed evosuite_test per ogni livello	Nelle cartelle dei primi 3 livelli sono stati inseriti gli output	PASS
6	Misurazione copertura utente	La classe e i test si trovano nel percorso specificato nell'input	/percorso_classe/+ /percorso_test/+ /percorso_salvataggio/	Statistic.csv	Nella cartella *report* è stato inserito l'output	PASS
7	Misurazione copertura utente	La classe non si trova nel percorso specificato nell'input	/percorso_classe_errato/+ /percorso_test/+ /percorso_salvataggio/	Messaggio di errore	Nella cartella *report* non è stato inserito l'output	PASS
8	Misurazione copertura utente	I test non si trovano nel percorso specificato nell'input	/percorso_classe/+ /percorso_test_errato/+ /percorso_salvataggio/	Messaggio di errore	Nella cartella *report* non è stato inserito l'output	PASS
9	Misurazione copertura utente	Il percorso di salvataggio esiste	/percorso_classe/+ /percorso_test/+ /percorso_salvataggio/	Statistics.csv	Nella cartella *report* è stato inserito l'output	PASS
10	Misurazione copertura utente	Il percorso di salvataggio non esiste	/percorso_classe/+ /percorso_test/+ /percorso_salvataggio_errato/	Messaggio di errore	Nella cartella *report* non è stato inserito l'output	PASS

12.2 Analisi tempo

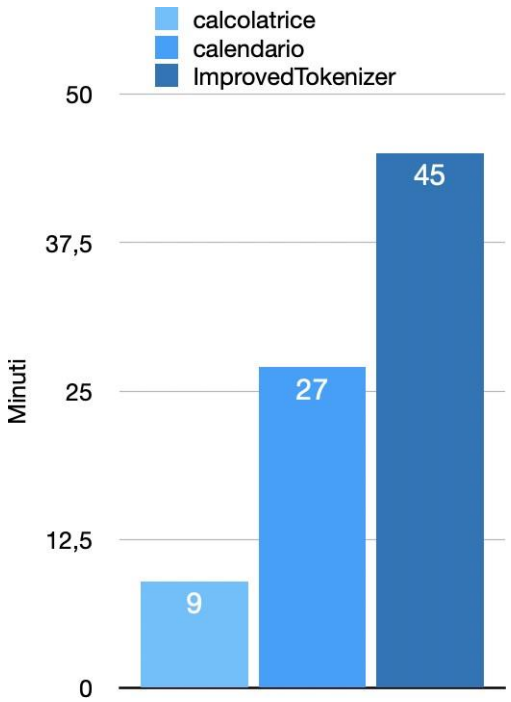


Figura 29: Analisi Tempo

Nella figura, sono mostrati i tempi di esecuzione della generazione dei livelli sul nostro computer. Per ogni livello in più, si aggiungono 1 o 2 minuti in più al tempo. Questi tempi posso variare in base alla potenza di calcolo del PC. Abbiamo sperimentato che su calcolatori più veloci, il tempo impiegato potrebbe ridursi significativamente.

Per quanto riguarda la misurazione, i tempi sono più costanti e pari all'incirca ad un minuto.

13. Guida all'utilizzo

Questo capitolo fornisce un'esaustiva panoramica sul corretto utilizzo del sistema. Alla fine di questo capitolo si avrà una comprensione completa su come installare, configurare ed utilizzare il sistema.

13.1 Dipendenze

Questo paragrafo ha il solo scopo di illustrare i componenti utilizzati per la realizzazione del software e non fornisce ancora una guida all'installazione. Di seguito, riportiamo le dipendenze comuni ad entrambi i sistemi:

- **OpenJDK-8**
- **Java** versione 1.8
- **Evosuite** versione 1.0.6
- **JUnit** versione 4.13.1
- **hamcrest-core** versione 1.3

In particolare, per la parte di misurazione utente, sarà necessario includere le seguenti dipendenze:

- **Nodejs** versione 19
- **npm** versione 6

Il sistema può essere eseguito sia su un computer con sistema operativo ubuntu 22.04 sia su container ubuntu docker.

13.2 Installazione

1. Prima di iniziare bisogna eseguire lo script *installazione.sh* che contiene i software necessari, come Evosuite, Maven e Nodejs

2. In opt/Prototipo2.0 sono presenti: i file ".js" per la creazione del server e uno script per la misurazione dei test scritti dall'utente

3. In opt_livelli sono presenti: lo script per la generazione dei test da parte di evosuite (*robot_generazione.sh*), il java che seleziona le iterazioni da utilizzare (*generazione_livelli.java*) e lo script per salvare le misurazioni e i test scelti nel repository comune (*misurazione_livelli.sh*)

13.3 Configurazione

1. Per avviare il server per la misurazione utente bisogna lanciare a linea di comando *node prova_esecuzione_parametri4.js*

2. Per avviare il server per la generazione dei livelli bisogna lanciare a linea di comando *robot_generazione:*

   2.1 Per un corretto avvio della generazione dei test e dei livelli bisogna lanciare lo script robot_generazione.sh nel seguente modo:
   >bash robot_generazione.sh NOME_CLASSE NOME_PACKAGE PERCORSO_PACKAGE NUMERO_LIVELLI
   esempio: >bash robot_generazione.sh calcolatrice calcolatrice mnt/f/Desktop/repository/calcolatrice 3

13.4 Rotta API

Le API (Application Programming Interface) sono un insieme di strumenti che consentono agli utenti del nostro sistema di interagire con esso in modo programmato, tramite scambio di dati e richieste. Le API sono fondamentali per la nostra architettura software, poiché consentono una maggiore flessibilità e scalabilità del sistema.

Le API sono progettate per essere modulari e consentono un facile accesso ai dati attraverso richieste HTTP.

Postman è un'applicazione per il testing di API che consente agli sviluppatori di creare, testare e documentare le loro API in modo rapido ed efficiente.

Di seguito, viene riportato il formato della richiesta http:

http://ip:porta/api/percorso_classe+percorso_test+percorso_salvataggio

- ip:porta: IP e porta su cui è connesso e comunica il server. Il numero della porta scelta è stato 3000.
- Percorso_classe: Percorso della classe da testare. In questo percorso deve essere presente sia il nome della classe che il nome del package in cui essa è contenuta.
- Percorso_test: Percorso della suite di test
- Percorso_salvataggio: Percorso della cartella in cui salvare i risultati ottenuti. Il nome del file, che verrà salvato, sarà “statistics.csv”.

Esempio di richiesta:

http://127.0.1.1:3000/api/data/StudentLogin/GameId/calcolatrice/calcolatrice.java+/data/StudentLogin/GameId/TestSourceCode/calcolatrice_test.java+/data/StudentLogin/GameId/TestReport

Viene simulata la richiesta http, sopra riportata, con Postman e viene illustrata, nella figura sottostante, la risposta:

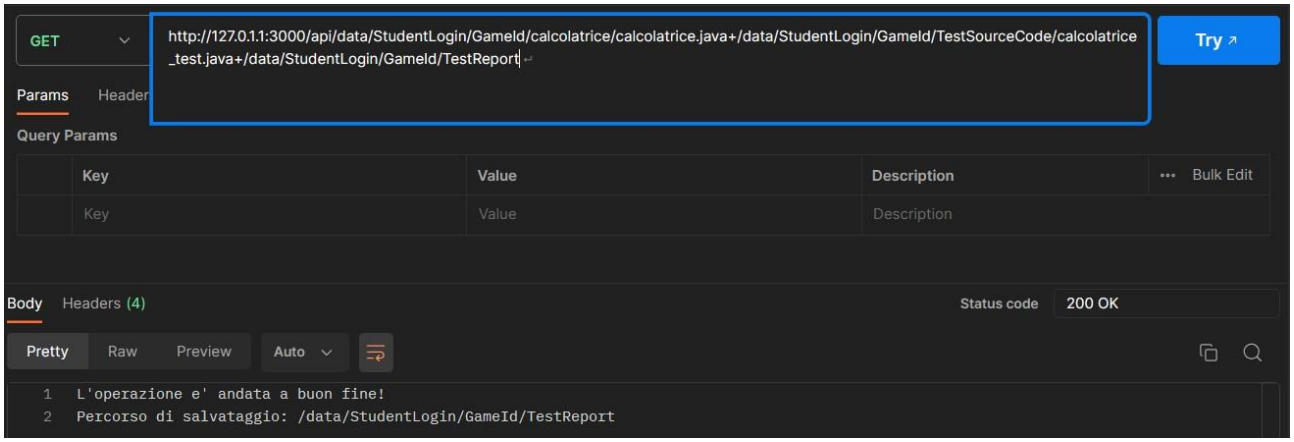


Figura 30: Postman

Per ulteriore chiarezza, la seguente tabella indica la descrizione, l'URL, il metodo, la risposta ed eventuali errori:

DESCRIZIONE:	L'utente richiede la misurazione della copertura della suite di test da lui indicata
URL:	*http://ip:porta/api/percorso_classe+percorso_test+percorso_salvataggio*
METODO:	GET
RISPOSTA:	L'operazione è andata a buon fine! Percorso di salvataggio: /percorso_salvataggio
ERRORI:	Timeout