



UNIVERSITÀ^{DEGLI STUDI DI}
NAPOLI FEDERICO II

Scuola Politecnica e delle Scienze di Base
Corso di Laurea Magistrale in Ingegneria Informatica

Software Architecture Design

Documentazione Progetto SAD

Anno Accademico 2023/2024

Professoressa: **Prof.ssa Anna Rita Fasolino**

Membri del Team:

Alberto Petillo M63001604

Benedetta Gaia Varriale M63001603

Francesco Altiero M63001609

Luigi Guerrera M63001386

Maria Concetta Arena N46005704

Raffaele Imperato M63001661

Contents

1	Introduzione	2
1.1	Descrizione del progetto	3
1.2	Descrizione dei requisiti assegnati	3
1.2.1	Requisito R4	3
1.2.2	Requisito R6	4
1.2.3	Suddivisione del lavoro in Task	4
1.3	Stato iniziale del progetto precedente alle modifiche	5
2	Metodologie di sviluppo	10
2.1	Daily Scrum	10
2.2	Sprint Backlog	11
2.3	Strumenti Utilizzati	11
2.3.1	Microsoft Teams	11
2.3.2	MiroBoard	11
2.3.3	Notion	12
2.3.4	GitHub	12
2.3.5	Visual Paradigm	13
2.4	Tecnologie e ambiente di sviluppo	13
2.4.1	Visual Studio Code	13
2.4.2	Spring Boot	14
2.4.3	Bootstrap	14

CONTENTS

2.4.4	Estensioni - Spring Boot Tools e Spring Boot Dashboard	15
2.4.5	Maven	16
3	Requisito R4	17
3.1	Analisi preliminare e refactoring	17
3.1.1	uploadClasse	18
3.1.2	uploadFile	21
3.1.3	FileUploadUtil.java	23
3.1.4	RobotUtil.java	25
3.2	1. Documentazione aggiornata	33
3.2.1	User Story	34
3.2.2	Use Case Diagram di T1 aggiornato	35
3.2.3	Sequence Diagram	36
3.3	2. Refactoring del metodo generateAndSaveRobots	37
3.3.1	A. Caricamento della classe nel Filesystem	40
3.3.2	B. Generazione dei test: lettura dell'output del processo	41
3.3.3	C. Salvataggio dei dati nel Task T4	42
3.4	3. Struttura dei file prodotti dai robot:	43
3.5	4. Modifiche alla pagina home_adm:	45
3.6	5. Pagina uploadClasseAndTest:	46
3.7	6. Modifiche al controller:	48
3.7.1	default.conf:	48
3.7.2	HomeController.java:	48
3.8	7. Metodo saveRobots:	50
3.9	Testing	55
3.10	Considerazioni finali	56
4	Requisito R6	57
4.1	User story	57
4.1.1	Installazione di Maven	58

CONTENTS

4.2	HomeController.java (T1)	60
4.3	Controller.java (T23) e GuiController.java (T5)	65
4.4	Altre risorse modificate	66
4.4.1	application.properties	66
4.4.2	default.conf	66
4.5	Soluzione alternativa 1	69
4.6	Soluzione alternativa 2	73
5	Organizzazione del Lavoro con Scrum	74
5.1	Introduzione a Scrum	74
5.1.1	Adozione di Scrum nel Progetto	74
5.2	User Stories Totali	75
5.2.1	Paradigma delle 3C	75
5.2.2	Criteri di Accettazione	76
5.3	Prima iterazione	76
5.4	Seconda iterazione	76
5.5	Terza iterazione	79
6	Guida all'installazione	82
6.1	Docker e WSL	82
6.2	Installazione	83
6.2.1	Passo 1	83
6.2.2	Passo 2	83
6.2.3	Passo 3	84
6.2.4	Utilizzo	84
7	Glossario	86
7.1	Robot	86
7.2	Turno	86
7.3	Giocatore	86

CONTENTS

7.4 Admin	87
7.5 Docker	87
7.6 Notion	87
7.7 Miro	87

CONTENTS

Il presente documento è fondamentale per tracciare lo sviluppo del progetto assegnato ai team A1 e A8, con il compito di implementare i requisiti R4 ed R6. Dall'analisi delle storie utente alla compilazione del codice, ogni fase, incluso l'avanzamento nella progettazione di casi d'uso e scenari, sarà accuratamente documentata.

Con l'implementazione di una metodologia AGILE, il processo di sviluppo si svolgerà in modo incrementale, facilitando una progettazione e implementazione progressiva. Particolare attenzione sarà rivolta agli aggiornamenti e alle ottimizzazioni dell'architettura software, garantendo che il risultato finale rifletta una struttura robusta. L'approccio architettonico sarà determinante per assicurare coerenza, flessibilità e scalabilità nel corso delle iterazioni del progetto.

Chapter 1

Introduzione

Il progetto European iNnovative AllianCe for TESTing, ENACTE-ST, si propone di enfatizzare l'importanza cruciale del testing nello sviluppo e nell'implementazione di applicazioni web. In diverse circostanze, si manifestano vulnerabilità ancora non identificate durante le fasi precedenti al lancio, evidenziando la necessità di affrontare tali questioni in modo efficace durante il processo di testing.

Nonostante ciò, il testing rimane spesso una disciplina trascurata e sottostimata all'interno del contesto dello sviluppo software. Al fine di superare questa sfida, l'Università di Napoli Federico II si impegna attivamente in collaborazione con piccole imprese per promuovere e consolidare l'importanza del testing.

L'approccio innovativo di ENACTE-ST si concretizza attraverso la creazione di un Educational Game. In tale contesto, i partecipanti sono chiamati a sfidare un software di generazione automatica di test, con l'obiettivo di coprire in modo esaustivo tutti i possibili scenari. Questo non solo offre un'opportunità pratica per lo sviluppo delle competenze nel testing, ma rende anche l'apprendimento di questa disciplina più coinvolgente e stimolante.

Attraverso ENACTE-ST, si auspica di incentivare una maggiore attenzione e valorizzazione per la disciplina del testing, contribuendo in modo significativo a garantire la solidità, la sicurezza e l'affidabilità delle applicazioni web.

1.1 Descrizione del progetto

Man vs Automated Testing Tools Challenges" costituisce un gioco educativo nel quale gli studenti si confrontano con strumenti di testing automatizzati capaci di generare automaticamente casi di test JUnit, come ad esempio Randoop o EvoSuite. L'obiettivo primario consiste nel raggiungere un determinato livello di copertura, quale ad esempio la copertura delle linee di codice (LOC).

Sul sito GitHub <https://github.com/Testing-Game-SAD-2023> è disponibile il lavoro di ciascun gruppo, ognuno incaricato di implementare requisiti funzionali accorpati in task. Per un'analisi più approfondita dei singoli task, si consiglia di consultare la documentazione specifica messa a disposizione da ciascun gruppo. In particolare per l'implementazione dei requisiti R4 ed R6, è stata consultata la documentazione relativa ai task T1 e T11 disponibili nella repository Github. L'implementazione è stata poi caricata in una fork della repository di <https://github.com/Testing-Game-SAD-2023/A1-2024>: <https://github.com/exo404/A1-A8-2024>. Nella cartella del task T1, all'interno della cartella "documentazione" è disponibile la documentazione insieme a un video dimostrativo del nuovo requisito e la classe di test usata per quest'ultimo.

1.2 Descrizione dei requisiti assegnati

1.2.1 Requisito R4

- Il Task T1, che consente all'amministratore di caricare una nuova classe da testare, interagisce con T8 e T9 per richiedere la generazione dei test ai 2 Robot. Tali

test sono poi salvati nei 1 volumi condivisi T8 e T9. T1 ha successivamente la responsabilità di salvare nel database di T4 i dati di sintesi sulla classe, sui Robot disponibili per essa, e i relativi livelli dei Test disponibili per ogni Robot. Per fare ciò T1 analizza il File system e deduce tali informazioni, per poi salvarle in T4.

- Verificare la dinamica di questo comportamento di T1 e prevederne una variante (scenario alternativo) che consenta di pre-caricare nel file system condiviso i Test dei Robot che siano già stati generati e di salvare in T4 i dati di sintesi di questi test.

1.2.2 Requisito R6

Realizzare una versione della app con messaggi in lingua Inglese

1.2.3 Suddivisione del lavoro in Task

Premettendo che questo lavoro è stato preceduto da una revisione del lavoro dei nostri colleghi che hanno integrato i vari componenti dell’architettura globale e documentato tutto come pubblicato su github <https://github.com/Testing-Game-SAD-2023/T11-G41/tree/master>.

Il lavoro è stato inizialmente suddiviso nei seguenti task con corrispondenti stime di persone necessarie per portare ciascuno a completamento:

- Task per la prima iterazione:
 - Traduzione dei file **HTML** in inglese [1 persona]
 - Individuazione dei percorsi che restituiscono pagine **HTML** e modificare la struttura dei file system in modo da aggiungere e sostituire i percorsi del tipo **/oldRoute** con **/it/oldRoute** e **/en/oldRoute** [2 persone]
 - Refactoring del codice del componente T1, con particolare attenzione alla funzione critica di salvataggio e quelle a sua volta invocate da essa [3 persone]

- Task per la seconda iterazione:
 - Creare una nuova Interfaccia Utente per la funzionalità R4 [3 persone]
 - Implementazione della funzionalità R4 vera e propria, a partire dalla funzione analizzata all'iterazione 1 [3 persone]

1.3 Stato iniziale del progetto precedente alle modifiche

Per comprendere al meglio il lavoro da noi svolto è opportuno mettere il punto su ciò che abbiamo avuto in eredità dai nostri colleghi che ci hanno lavorato in precedenza. Prima di fare qualsiasi considerazione sul progetto in sé è necessario soffermarsi su un aspetto fondamentale: l'installazione del programma. È stata la prima cosa da fare quando ci siamo messi all'opera e fin da subito abbiamo notato che le guide già presenti non erano del tutto esaustive. Per questo motivo, abbiamo riscontrato non poche difficoltà in questa fase preliminare che fortunatamente siamo riusciti a risolvere. Onde evitare che ciò succeda anche con i futuri fruitori del programma abbiamo redatto una **nuova guida all'installazione** passo dopo passo. È possibile consultarla nel **capitolo 6**.

Il sistema su cui siamo andati ad apportare le modifiche richieste presentava la seguente struttura componenti e connettori:

CHAPTER 1. INTRODUZIONE

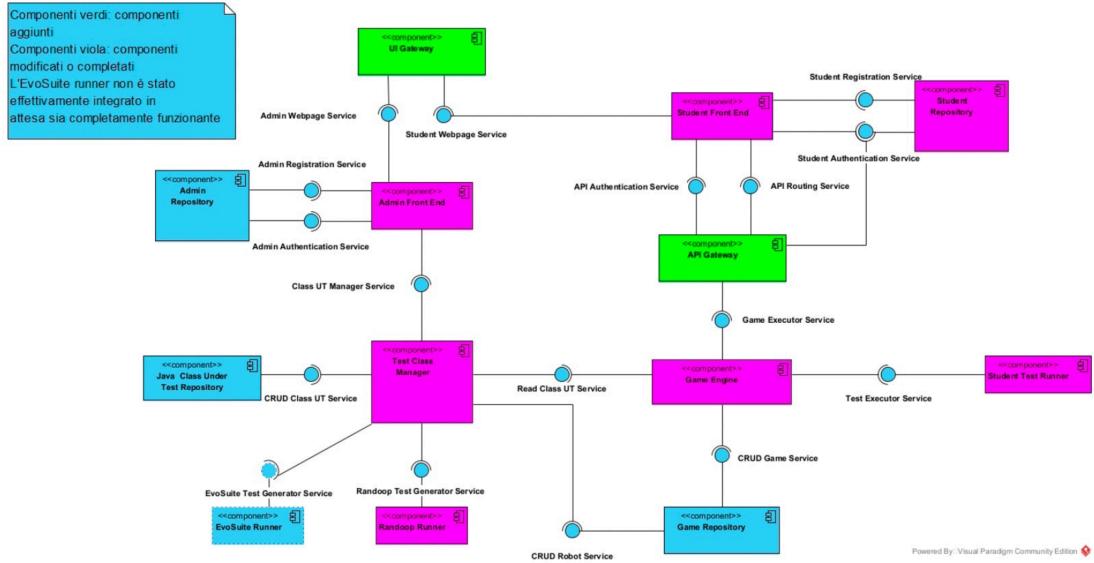


Figure 1.1: vista componenti e connettori

Relativamente al diagramma appena mostrato, noi lavoriamo, per quanto concerne al requisito R4, all'interno dell'**Admin Front End** e nel **Test Class Manager**, e per il requisito R6 invece anche **Student Front End**. In particolare il requisito R4 è relativo al **task T1** di cui abbiamo studiato attentamente la documentazione durante la prima iterazione. Il task T1 presenta la seguente vista componenti e connettori.

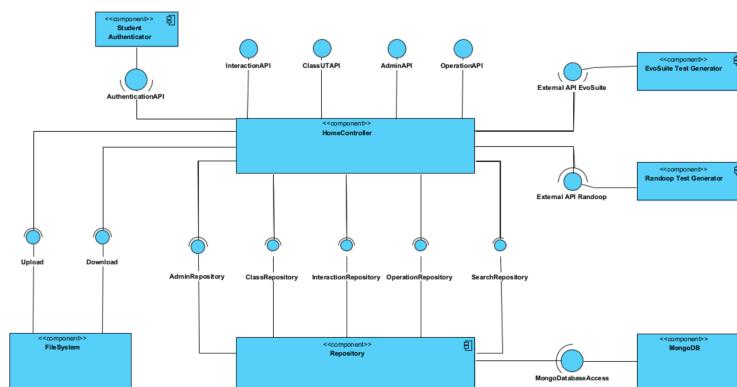


Figure 1.2: vista componenti e connettori T1

CHAPTER 1. INTRODUZIONE

All'interno di questa vista noi lavoriamo principalmente con l'**HomeController** e il **FileSystem**. Per maggiore chiarezza mostriamo anche i diagrammi di package e classi del T1.

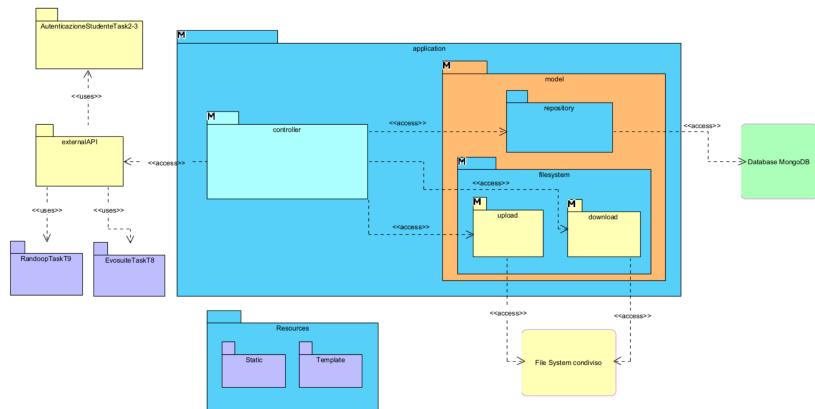


Figure 1.3: diagramma dei package iniziale

Il diagramma delle classi è mostrato nella seguente pagina:

CHAPTER 1. INTRODUZIONE

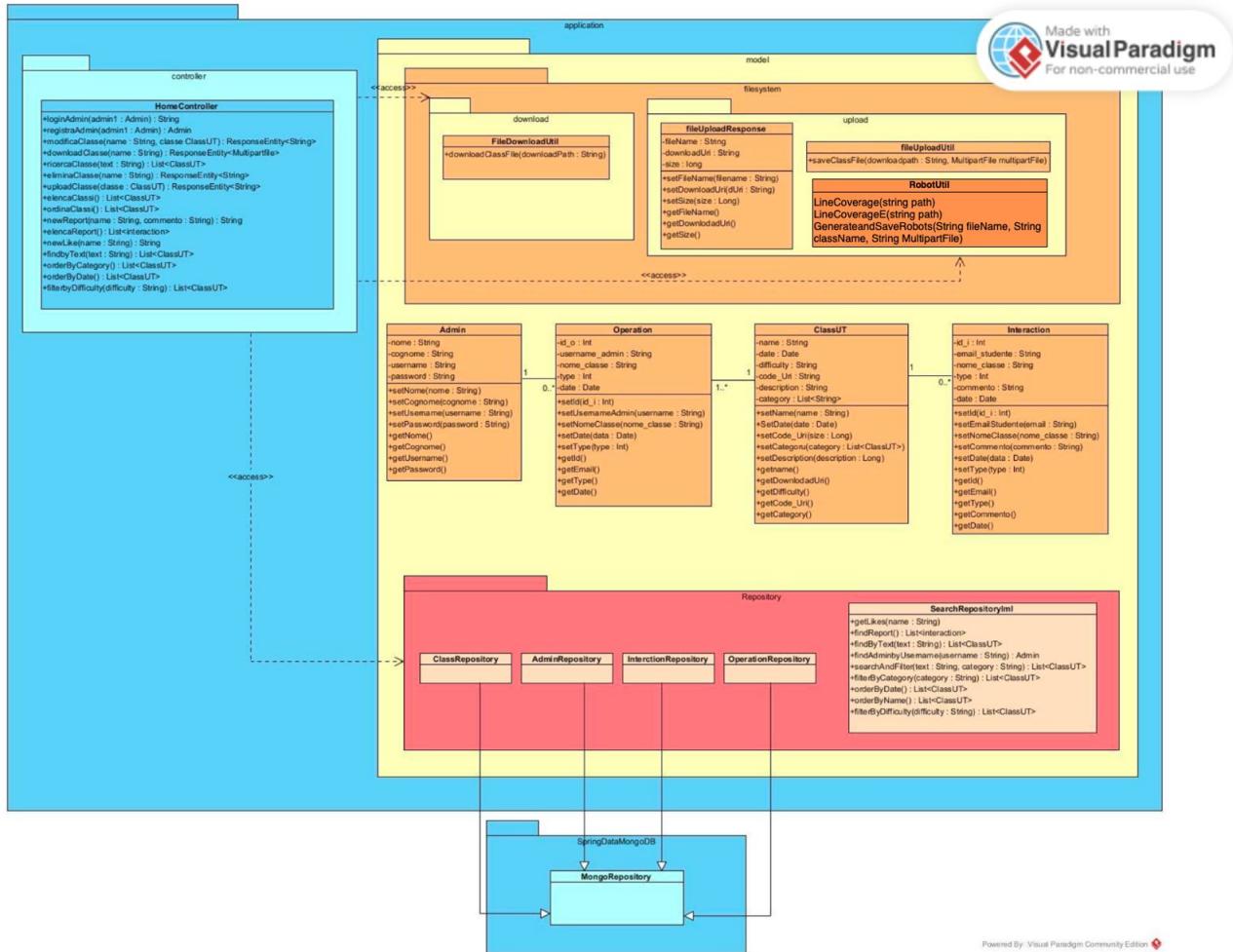


Figure 1.4: diagramma delle classi

Dalla nostra analisi è risultato che il nodo centrale da analizzare (e in seguito modificare) era quello riguardante il metodo *UploadClasse*. Nel **capitolo 3** analizzeremo meglio il diagramma di sequenza iniziale.

Per comprendere al meglio dove fosse necessario apportare la modifica a livello di codice abbiamo analizzato i codice della repository notando che spesso erano privi di commenti e poco chiari in determinati passaggi. Ciò ha reso ovviamente più lento il processo di

piena comprensione del funzionamento iniziale. Per questo motivo, prima di iniziare a modificare, abbiamo prestato particolare attenzione alla lettura e comprensione del codice concentrandoci maggiormente sui metodi che avremmo dovuto utilizzare per procedere con il nostro lavoro. Durante questo processo di comprensione è stato effettuato un refactoring (anzì due, uno durante la prima fase relativa alla seconda iterazione e un secondo durante la terza iterazione). Abbiamo commentato riga per riga il codice e reso più chiari e lineari determinati passaggi, modificando anche nomi di variabili da noi ritenuti un po' ambigui e non abbastanza descrittivi (ad esempio, la variabile *cname* indicante il nome della classe è stata rinominata in *className*). In particolare, utilizzando un approccio che potrebbe essere classificato come utilizzo del **pattern extract method** abbiamo snellito la scrittura della funzione **generateAndSaveRobot** estraendo dal metodo originale altri metodi che verranno chiamati più volte. In questo modo il tutto risulta più lineare e di facile comprensione. Anche questo aspetto verrà analizzato nel dettaglio nel capitolo 3. Nel capitolo 4 viene poi presentata una soluzione per il requisito R4 in cui vengono modificati, oltre ai file html che devono essere tradotti, anche dei file di configurazione globali ed i controller di T1, T5 e T23 in modo da servire le richieste in base alla lingua, senza compromettere le funzionalità. Sono poi presentate delle soluzioni alternative in quanto, durante una Sprint Review, sembrava che la prima soluzione non fosse capace di essere integrata facilmente con il lavoro relativo agli altri requisiti.

Chapter 2

Metodologie di sviluppo

Il primo passo per la realizzazione del task assegnato è stato quello di introdurre la metodologia **AGILE** basata su **Daily SCRUM** e **Sprint Backlog**. La riunione iniziale del team ha avuto come obiettivo proprio l'identificazione dello Sprint Backlog, analizzando preliminarmente la traccia assegnata per gestire eventuali scadenze

2.1 Daily Scrum

La prima e principale difficoltà è stata riscontrata sicuramente nell'analizzare il dominio del problema da affrontare. Per portare a termine questo processo di analisi sono state organizzate varie riunioni con cadenza variabile, giornaliere o con cadenza max 3 giorni, per dare una prima forma vera e propria di organizzazione al team. Gli incontri si sono tenuti sia in presenza, presso l'università, sia da remoto attraverso l'utilizzo di Microsoft Teams. Le riunioni in presenza sono state utilizzate principalmente per discutere delle problematiche principali che ogni componente del gruppo riscontrava, mentre le riunioni da remoto sono state utilizzate principalmente per gestire e organizzare tutto il lavoro che doveva essere svolto.

2.2 Sprint Backlog

Le riunioni organizzative hanno rappresentato un fondamentale strumento per definire concretamente gli Sprint Backlog. Durante tali incontri, si è proceduto a una prima assegnazione dei task tra i membri del team e all'individuazione di scadenze per il conseguimento degli obiettivi.

In particolare, nel periodo compreso tra ottobre e dicembre, le deadline sono state inizialmente fissate ogni due settimane. Con l'avvicinarsi della data di presentazione del progetto, è emersa la necessità di intensificare gli sforzi e incrementare la produttività. Di conseguenza, è stata adottata una frequenza settimanale per le scadenze, al fine di ottimizzare il lavoro e favorire il raffinamento degli artefatti in vista della presentazione finale.

2.3 Strumenti Utilizzati

2.3.1 Microsoft Teams

L'applicazione Microsoft Teams è risultata fondamentale per facilitare la comunicazione del team durante le fasi di sviluppo degli artefatti. Le videochiamate di gruppo hanno consentito discussioni dettagliate, mentre l'utilizzo della chat del canale dedicato ha semplificato lo scambio rapido di file e messaggi. La piattaforma ha contribuito a mantenere una collaborazione efficace, soprattutto in periodi di lavoro remoto.

2.3.2 MiroBoard

MiroBoard è uno strumento di lavagna digitale che ha agevolato la collaborazione remota del team. Le sue caratteristiche includono:

- **Piattaforma di Collaborazione Visiva:** Offre un ambiente virtuale per la creazione e la condivisione di lavagne digitali, facilitando attività come brainstorm-

ing, diagrammi, mappe concettuali e flussi di lavoro.

- **Strumenti di Collaborazione:** Fornisce una varietà di strumenti, inclusi pennarelli virtuali e forme, per supportare la collaborazione in tempo reale.
- **Integrazioni:** Si integra con diverse applicazioni e servizi, consentendo l'importazione di dati da strumenti come Jira, Trello e Slack.
- **Funzionalità di Presentazione:** Permette la creazione di presentazioni dinamiche direttamente sulla lavagna.
- **Sicurezza e Collaborazione Sicura:** Offre funzionalità di sicurezza per proteggere dati sensibili e controllare l'accesso alle lavagne.

2.3.3 Notion

Notion è uno strumento di produttività che ha agevolato la collaborazione remota del team. Fornisce una varietà di strumenti per supportare la collaborazione in tempo reale.

2.3.4 GitHub

GitHub, basato su Git, è una piattaforma di sviluppo collaborativo. Le sue caratteristiche principali includono:

- **Controllo Versione Distribuito:** Utilizza Git come sistema di controllo versione distribuito per tenere traccia delle modifiche nel codice sorgente.
- **Repository:** Ospita repository Git che contengono file e la cronologia delle revisioni del progetto software.
- **Collaborazione:** Agevola la collaborazione consentendo a più sviluppatori di lavorare contemporaneamente, contribuendo al codice e gestendo problemi e richieste di pull.

2.3.5 Visual Paradigm

Visual Paradigm, uno strumento di modellazione, ha svolto un ruolo chiave nella fase di progettazione e definizione dei requisiti. Ha facilitato la creazione di vari diagrammi UML, come parte integrante della documentazione del progetto.

2.4 Tecnologie e ambiente di sviluppo

Come menzionato nella sezione precedente, è stato impiegato l'editor **Visual Studio Code** per la scrittura e la compilazione del codice. Pur nascendo come editor di codice leggero, l'intero team se n'è servito utilizzandolo come ambiente di sviluppo, in quanto è considerabile un IDE adeguato grazie alla possibilità di aggiungere delle estensioni per l'introduzione di framework e altre tecnologie.

2.4.1 Visual Studio Code

Visual Studio Code è un editor di codice sorgente leggero e gratuito. Esso è stato utilizzato da tutto il team di sviluppo per scrivere e compilare eventuali modifiche effettuate all'interno del codice. La compilazione del nuovo codice è stata possibile grazie anche all'utilizzo di diverse estensioni per ottenere il risultato richiesto. Come primo framework particolarmente importante da menzionare c'è sicuramente **SpringBoot**.

- **Leggerezza e Velocità:** Design leggero e avvio rapido anche su macchine meno potenti.
- **Supporto Multi-linguaggio:** Ampia gamma di estensioni per il supporto di vari linguaggi di programmazione.
- **Integrazione con Git:** Offre un'integrazione completa con Git, permettendo operazioni di versionamento direttamente dall'editor.

2.4.2 Spring Boot

Spring Boot rappresenta un framework open-source basato su Java progettato per semplificare il processo di sviluppo di applicazioni Java. La sua filosofia si concentra sull'offrire un'infrastruttura pronta all'uso per affrontare le complessità comuni nello sviluppo software. Ciò consente agli sviluppatori di dedicare maggior tempo alla logica aziendale centrale anziché alle configurazioni e alle implementazioni tecniche.

Le caratteristiche chiave di Spring Boot includono la gestione delle dipendenze, la configurazione automatica, la possibilità di integrare server Web incorporati e una facilità di sviluppo e test. Questo framework è ampiamente apprezzato per la sua capacità di semplificare la creazione di applicazioni Java, offrendo scalabilità, robustezza e una gestione agevole. In sostanza, Spring Boot è una scelta diffusa per lo sviluppo rapido di applicazioni Java che devono essere facilmente scalabili e gestibili.

2.4.3 Bootstrap

Bootstrap rappresenta un framework front-end open-source concepito per semplificare il processo di progettazione e sviluppo di siti web responsivi e intuitivi. Questo strumento mette a disposizione una ricca collezione di componenti predefiniti, tra cui pulsanti, moduli, barre di navigazione e sistemi di layout flessibili, agevolando la rapida costruzione di interfacce utente moderne e esteticamente gradevoli.

Utilizzando una combinazione di HTML, CSS e JavaScript, Bootstrap fornisce un insieme coerente di strumenti e stili che facilitano la creazione di siti web responsive, adattabili in modo automatico a diverse dimensioni di schermi e dispositivi. La flessibilità di Bootstrap emerge anche dalla sua elevata personalizzabilità e dalla disponibilità di una vasta gamma di temi e plugin aggiuntivi, permettendo agli sviluppatori di estendere le funzionalità di base in modo semplice.

Nel campo dello sviluppo web, Bootstrap è ampiamente adottato per ottimizzare il tempo

e gli sforzi dedicati alla creazione di interfacce utente di alta qualità. In sintesi, Bootstrap rappresenta una risorsa essenziale che contribuisce alla realizzazione di progetti web avanzati e all'avanguardia.

2.4.4 Estensioni - Spring Boot Tools e Spring Boot Dashboard

Gli strumenti **Spring Boot**, inclusi **Spring Boot Tools** e **Spring Boot Dashboard**, costituiscono un insieme prezioso di risorse per semplificare e potenziare il processo di sviluppo delle applicazioni Spring Boot all'interno dell'ambiente di Visual Studio Code. Questi strumenti sono progettati per ottimizzare l'efficienza e la produttività degli sviluppatori.

Spring Boot Tools si distingue per le sue funzionalità avanzate, tra cui il completamento automatico del codice, l'analisi statica, il debug e la gestione delle dipendenze specifiche di Spring Boot. Un vantaggio significativo è la capacità di eseguire direttamente le applicazioni Spring Boot da Visual Studio Code, semplificando il ciclo di sviluppo e testing.

Dall'altro lato, Spring Boot Dashboard offre un'interfaccia intuitiva per visualizzare e gestire le applicazioni Spring Boot in esecuzione sia localmente che su server remoti. Le sue funzioni dettagliate includono la visualizzazione dello stato delle applicazioni, la possibilità di avviare, arrestare e riavviare le applicazioni, oltre a fornire un accesso immediato ai log dell'applicazione e alle metriche di monitoraggio delle prestazioni.

In sintesi, questi strumenti forniscono una suite completa per ottimizzare il flusso di lavoro di sviluppo in ambiente Spring Boot, garantendo un ambiente integrato e efficiente all'interno di Visual Studio Code.

2.4.5 Maven

Maven è uno strumento di gestione di progetti Java progettato per semplificare diverse fasi del ciclo di vita dello sviluppo. Una delle sue caratteristiche principali è la gestione agevole delle dipendenze e il processo di compilazione. Attraverso il file di configurazione "pom.xml", è possibile specificare le librerie necessarie per il progetto, e Maven si occupa automaticamente di scaricarle dalla repository remota.

Inoltre, Maven promuove l'adozione di una struttura di progetto standardizzata, facilitando la collaborazione all'interno del team. Grazie al suo ciclo di vita predefinito, è possibile eseguire facilmente operazioni come la compilazione, i test e la creazione del pacchetto dell'applicazione.

Un aspetto importante di Maven è la sua estensibilità attraverso l'uso di plugin. Questi plugin consentono di ampliare le funzionalità di base di Maven, come il controllo della qualità del codice o la generazione automatica della documentazione. In sostanza, Maven fornisce un ambiente organizzato e automatizzato per lo sviluppo di progetti Java, migliorando l'efficienza e la coerenza nel processo di sviluppo del software.

Chapter 3

Requisito R4

3.1 Analisi preliminare e refactoring

Per prepararci alla terza iterazione in cui verrà affrontato il requisito R4, abbiamo eseguito un refactoring del codice interessato dalle prossime modifiche, modificando il nome di alcune variabili ritenuti da noi poco esplicativi e commentando il codice in modo da comprendere al meglio la catena di chiamate che rendono possibile il caricamento di una nuova classe all'interno dell'applicazione. Per cominciare è stato rianalizzato il diagramma di sequenza della funzionalità uploadClasse, del task T4. Il diagramma di sequenza da cui siamo partiti, presente nella documentazione redatta dai nostri colleghi dell'anno precedente viene mostrato nella prossima pagina: **Figure 3.1**

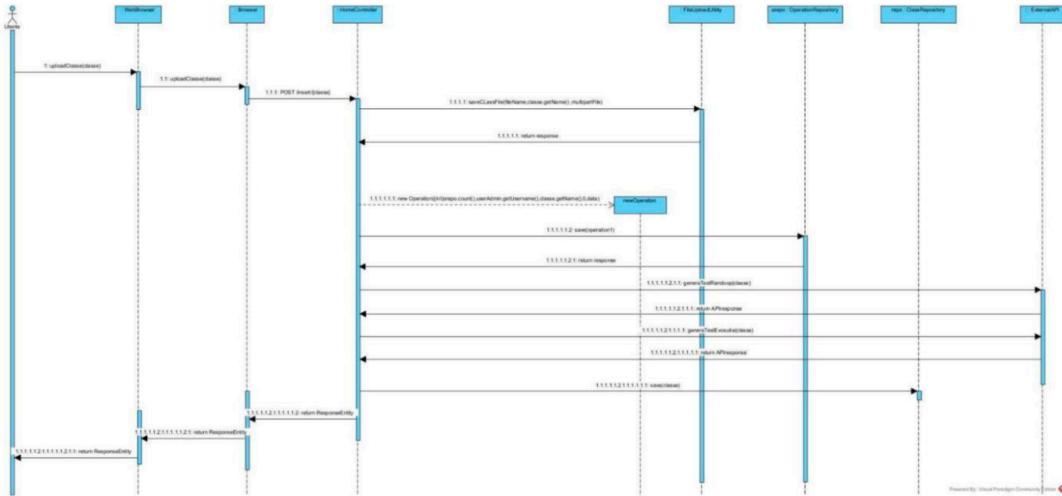


Figure 3.1: diagramma di sequenza task R4

Per apportare le modifiche richieste andiamo a lavorare in due parti distinte dell’architettura, sia controller che model. In seguito riportiamo le parti del codice interessate dal refactoring seguendo l’ordine delle chiamate nel diagramma di sequenza.

3.1.1 uploadClasse

È un metodo della classe **HomeController** appartenente al package **controller**. Adesso andiamo ad analizzare la parte in html ovvero ciò che l’utente (in questo caso l’amministratore che desidera caricare la classe) vede dal browser. Il codice html che stiamo andando ad analizzare fa parte delle risorse a disposizione. Si trova nel package **resources** all’interno del package **template**. **Figure 3.2**

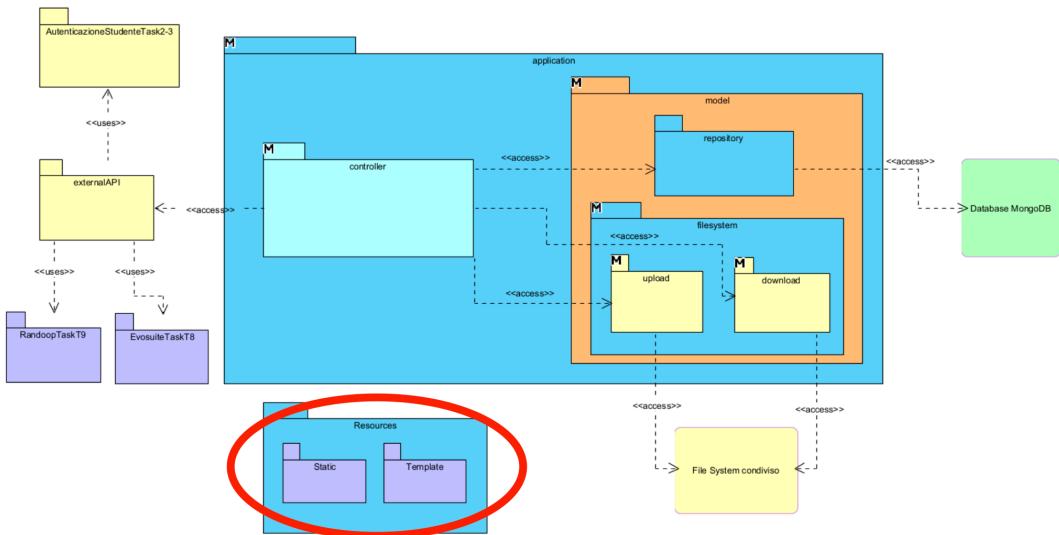


Figure 3.2: posizione di homecontroller.html

Alla fine del file HTML è presente uno script Javascript che viene chiamato alla fine della compilazione del form presente nella pagina HTML, ogni volta che viene cliccato il tasto upload in verde. Nella prima parte vengono inizializzate delle costanti che rappresentano gli input del form **Figure 3.3**

```

<script>
//Si aggancia al form tramite ID
const form = document.getElementById('formId');

//Creazione delle costanti che rappresentano i dati input al form
const name = document.getElementById('className').value;
const date = document.getElementById('date').value;
const difficulty = document.getElementById('difficulty').value;
const code_Uri = "" ;
const description = document.getElementById('description').value;
const category = [
    document.getElementById('category1').value,
    document.getElementById('category2').value,
    document.getElementById('category3').value
];

//Costante che rappresenta il file .java in input
const classInput = document.getElementById('fileInput');
const file = classInput.files[0];

```

Figure 3.3: homecontroller.html

Viene poi preparato il body del messaggio HTML impacchettando i dati del form e il loro modello all'interno di un'unica costante.

Figure 3.4

```
//Costante che rappresenta l'insieme dei dati del form e il file in input
const formData = new FormData();
formData.append('file', file);
formData.append('model', JSON.stringify({
    name: name,
    date: date,
    difficulty: difficulty,
    code_Url: code_Url,
    description: description,
    category: category
}));
```

Figure 3.4: preparazione body messaggio HTML

Viene quindi richiamata la funzione uploadFile che si trova all'interno del **controller** all'interno della classe **HomeController** con metodo POST e body appena preparato.

Di seguito sono stati lasciati dei commenti dai gruppi precedenti che non hanno gestito le risposte del server.

Figure 3.5

```
//chiamata http alla funzione uploadFile in HomeController con metodo POST e dati del form come body
fetch('/uploadFile', {
  method: 'POST',
  body: formData
})
.then(response => response.json())
.then(data => {
  console.log('Success:', data);
  window.location.href = "/home_adm";
  // Aggiungi qui il codice per gestire la risposta dal server
})
.catch((error) => {
  console.error('Error:', error);
  // Aggiungi qui il codice per gestire gli errori
});

```

Figure 3.5: chiamata funzione uploadfile

3.1.2 uploadFile

Successivamente si ha la chiamata al metodo **uploadFile**. Nella firma della funzione ritroviamo le due parti del body inviato dal front-end. Abbiamo rinominato la prima variabile da file a formData per richiamare la costante dichiarata nel front-end. Model non è stata modificata in quanto rappresenta effettivamente il modello dei dati. **Figure 3.6**

```
@PostMapping("/uploadFile")
@ResponseBody
public ResponseEntity<FileUploadResponse> uploadFile(@RequestParam("file") MultipartFile formData, @RequestParam("model") String
model) throws IOException {
```

Figure 3.6: Definizione upload file

Tramite la classe ObjectMapper vengono incapsulati i dati del form all'interno di un oggetto ClasseUT. **Figure 3.7**

```
//Legge i metadati della classe della parte "model" del body http e li salva in un oggetto ClasseUT
ObjectMapper mapper = new ObjectMapper();
ClasseUT classe = mapper.readValue(model, ClasseUT.class);
```

Figure 3.7: Lettura metadati

Di seguito viene estratto il nome del file caricato nel form, insieme alla sua grandezza e viene caricato nel filesystem condiviso attraverso la classe FileUploadUtil. **Figure 3.8**

Figure 3.9

```
//salva il nome del file caricato
String fileName = StringUtils.cleanPath(formData.getOriginalFilename());
long size = formData.getSize();
```

Figure 3.8: Salvataggio fileName

```
//Salva la classe nel filesystem condiviso
FileUploadUtil.saveClassFile(fileName, classe.getName(), formData);
```

Figure 3.9: Salvataggio Classe

Viene richiamata la funzione generateAndSaveRobots della classe RobotUtil (package **filesystem** interno al **model**) per la generazione e il salvataggio dei test dei robot nel file condiviso. Poi viene preparata la risposta di OK per il front-end. **Figure 3.10**

```
//Genera e salva i test nel filesystem condiviso
RobotUtil.generateAndSaveRobots(fileName, classe.getName(), formData);

FileUploadResponse response = new FileUploadResponse();
response.setFileName(fileName);
response.setSize(size);
response.setDownloadUri("/downloadFile");
```

Figure 3.10: Generazione e salvataggio Test nel FS

La funzione setta i valori tralasciati dal form ovvero data di caricamento e percorso di download della classe. **Figure 3.11**

```
//Setta data di caricamento e percorso di download
classe.setUri("Files-Upload/" + classe.getName() + "/" + fileName);
classe.setDate(today.toString());
```

Figure 3.11: Settaggio percorso di download

Viene creato un oggetto per la data corrente e un oggetto Operation che registra il caricamento della classe da parte dell'admin all'interno del database. **Figure 3.12**

```
//Creazione dell'oggetto riguardante l'operazione appena fatta
LocalDate currentDate = LocalDate.now();
DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd");
String data = currentDate.format(formatter);
Operation operation1 = new Operation((int) orepo.count(), userAdmin.getUsername(), classe.getName(), 0, data);
```

Figure 3.12: Creazione oggetto per la data

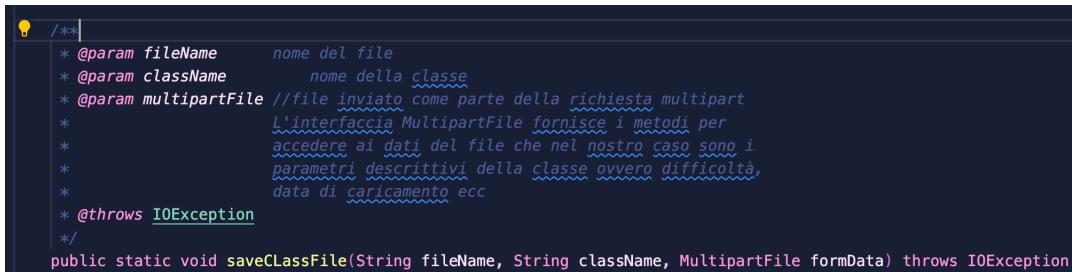
Infine, vengono salvate nel database sia l'operazione che le informazioni sulla classe. In questo passaggio abbiamo l'interazione con il task T4 (Lo specificheremo meglio nel diagramma di sequenza aggiornato) **Figure 3.13**

```
//Salva i dati sull'operazione fatta nel database
orepo.save(operation1);
//Salva i dati sulla classe nel database
repo.save(classe);
return new ResponseEntity<>(response, HttpStatus.OK);
```

Figure 3.13: salvataggio nel database

3.1.3 FileUploadUtil.java

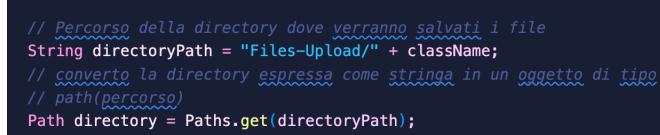
La classe FileUploadUtil si trova all'interno del package dell'**upload**, a sua volta contenuto nel package del **filesystem** che fa parte del **model**. Alla funzione saveClassFile per il salvataggio della classe nel filesystem condiviso vengono passati il nome del file .java, il nome della classe e il file caricato nel form. Anche qui sono stati cambiati: il nome della variabile file in formData e il nome della variabile cname in className. Nella seguente pagina vediamo la definizione di **saveClassFile**. **Figure 3.14**



```
/***
 * @param fileName      nome del file
 * @param className     nome della classe
 * @param multipartFile //file inviato come parte della richiesta multipart
 *                      L'interfaccia MultipartFile fornisce i metodi per
 *                      accedere ai dati del file che nel nostro caso sono i
 *                      parametri descrittivi della classe ovvero difficoltà,
 *                      data di caricamento ecc
 * @throws IOException
 */
public static void saveClassFile(String fileName, String className, MultipartFile formData) throws IOException
```

Figure 3.14: Definizione saveClassFile

Viene costruita una stringa che rappresenta il percorso della cartella in cui verrà caricata la classe e una oggetto Path che la incapsula. A questo punto viene creata la cartella con il percorso specificato. **Figure 3.15 Figure 3.16**



```
// Percorso della directory dove verranno salvati i file
String directoryPath = "Files-Upload/" + className;
// converto la directory espressa come stringa in un oggetto di tipo
// path(percuso)
Path directory = Paths.get(directoryPath);
```

Figure 3.15: Percorso della directory in cui saranno salvati i file



```
try {
    // Verifica se la directory esiste già
    if (!Files.exists(directory)) {
        // Crea la directory
        Files.createDirectories(directory);
        System.out.println("La directory è stata creata con successo.");
    } else {
        System.out.println("La directory esiste già.");
    }
} catch (Exception e) {
    System.out.println("Errore durante la creazione della directory: " + e.getMessage());
}
```

Figure 3.16: Creazione directory

Viene quindi copiato il file nella directory.

Figure 3.7

```
// Percorso completo della directory di upload
Path uploadDirectory = Paths.get("Files-Upload/" + className);

try (InputStream inputStream = formData.getInputStream()) {
    // Risolve il percorso del file all'interno della directory
    // ovvero viene creato un oggetto Path che rappresenta il percorso completo del
    // file all'interno della directory di destinazione
    // Il metodo resolve è utilizzato per ottenere il percorso completo concatenando
    // il percorso della directory (uploadDirectory) con il nome del file
    // (fileName).
    Path filePath = uploadDirectory.resolve(fileName);

    // Copia il file nell'uploadDirectory con opzione di sovrascrittura se esiste
    // già
    Files.copy(inputStream, filePath, StandardCopyOption.REPLACE_EXISTING);
}
```

Figure 3.17: Copia del file

3.1.4 RobotUtil.java

La classe **RobotUtil.java** si trova all'interno del package del **Filesystem** all'interno del **model**. Vediamo la definizione della funzione **generateAndSaveRobots**. I parametri in ingresso sono gli stessi della funzione precedente, come anche le modifiche apportate.

Vediamo la definizione della funzione generateAndSaveRobots: **Figure 3.18**

```
/**
 * Genera e salva i file di robot e ne calcola la copertura delle linee.
 *
 * @param fileName      Il nome del file.
 * @param className     L'ID della classe di test.
 * @param formData      Il file caricato.
 * @throws IOException  Eccezione di IO.
 */
public static void generateAndSaveRobots(String fileName, String className, MultipartFile formData)
    throws IOException {
```

Figure 3.18: definizione di generateAndSaveRobots

Viene creata la directory per il caricamento del file caricato nel form. Quest'ultimo poi viene copiato nella cartella. **Figure 3.19** **Figure 3.20** **Figure 3.21** **Figure 3.22**

```
try {
    // Verifica se la directory esiste già
    if (!Files.exists(directory)) {
        // Crea la directory
        Files.createDirectories(directory);
        System.out.println("La directory è stata creata con successo.");
    } else {
        System.out.println("La directory esiste già.");
    }
} catch (Exception e) {
    System.out.println("Errore durante la creazione della directory: " + e.getMessage());
}
```

Figure 3.19: creazione directory

```
// Legge l'input stream del file caricato e lo copia nella directory specificata
try (InputStream inputStream = formData.getInputStream()) {
```

Figure 3.20: lettura iostream

```
// Risolve il percorso completo del file all'interno della directory
// specificata.
// Viene utilizzato il metodo 'directory.resolve(fileName)' per ottenere il
// percorso completo
// del file all'interno della directory 'directory'. Questo percorso completo
// sarà utilizzato
// successivamente per copiare il contenuto dell'input stream del file nella
// posizione desiderata.
Path filePath = directory.resolve(fileName);
System.out.println(filePath.toString());
```

Figure 3.21: risoluzione directory

```
// copio il contenuto dell'input stream nel file di destinazione
// l'ultimo parametro di questa funzione indica che se il file già esiste deve
// essere sostituito
Files.copy(inputStream, filePath, StandardCopyOption.REPLACE_EXISTING);

// chiusura dell'input stream dopo aver completato la copia
inputStream.close();
```

Figure 3.22: copia e chiusura

Viene creato un oggetto Process attraverso il ProcessBuilder con cui viene impostato il comando per la generazione dei test Randoop e la directory di lavoro del processo. Il processo viene poi avviato. **Figure 3.23** **Figure 3.24** **Figure 3.25** **Figure 3.26**

```
// creazione del processo esterno
ProcessBuilder processBuilder = new ProcessBuilder();
```

Figure 3.23: creazione processo

```
// con command si configura il comando del processo esterno per eseguire il file
// JAR 'Task9-G19-0.0.1-SNAPSHOT.jar'
// l'esecuzione avviene attraverso la JVM di Java.
// Il parametro "-jar" specifica l'esecuzione di un file JAR.

processBuilder.command("java", "-jar", "Task9-G19-0.0.1-SNAPSHOT.jar");
```

Figure 3.24: configurazione processo

```
// La directory di lavoro per il processo esterno viene impostata su
// "/VolumeT9/app/" utilizzando
// questo metodo garantisce che il processo lavori nella directory desiderata
processBuilder.directory(new File("/VolumeT9/app/"));
```

Figure 3.25: settaggio directory

```
// si avvia il processo
Process process = processBuilder.start();
```

Figure 3.26: avvio processo

Il metodo legge l'output del processo e ne restituisce l'*exitCode*.

Figure 3.27 **Figure 3.28** **Figure 3.29** **Figure 3.30**

```
// Legge l'output del processo esterno tramite un BufferedReader, che a sua
// volta usa
// un InputStreamReader per convertire i byte in caratteri. Il metodo
// 'process.getInputStream()'
// restituisce lo stream di input del processo esterno.

BufferedReader reader = new BufferedReader(new InputStreamReader(process.getInputStream()));
String line;
```

Figure 3.27: lettura dell'output

```
// All'interno del loop viene letta ogni linea disponibile finché il processo
// continua a produrre output.
while ((line = reader.readLine()) != null)
    System.out.println(line);
```

Figure 3.28: ciclo while

```
// funzionamento analogo al precedente, invece di leggere l'output leggiamo gli
// errori
reader = new BufferedReader(new InputStreamReader(process.getErrorStream()));
while ((line = reader.readLine()) != null)
    System.out.println(line);
```

Figure 3.29: lettura errori

```
try {
    // Attende che il processo termini e restituisce il codice di uscita
    int exitCode = process.waitFor();

    System.out.println("ERRORE CODE: " + exitCode);
} catch (InterruptedException e) {
    System.out.println(e);
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

Figure 3.30: attesa terminazione

Di seguito vengono elaborati i risultati della generazione. Viene costruito il percorso dei risultati dei test attraverso il nome della classe, creato un array di file contenente tutti i file della directory e inizializzato il livello corrente a 0. **Figure 3.31 Figure 3.32**

```
// Crea un oggetto File che rappresenta il percorso della directory contenente i
// risultati
// della generazione di robot da Randoop. Il percorso è costruito in base all'ID
// della classe di test 'className'.
File resultsDir = new File("/VolumeT9/app/FolderTree/" + className + "/RobotTest/RandoopTest");
```

Figure 3.31: creazione directory risultati

```
// Inizializza la variabile 'liv' a 0, rappresentante il massimo livello di
// robot prodotti da Randoop.
// Questo valore sarà aggiornato successivamente durante l'analisi dei
// risultati.
int liv = 0; // livelli di robot prodotti da randoop

File results[] = resultsDir.listFiles();
// Itera attraverso tutti i file nella directory dei risultati della generazione
// di robot da Randoop.

for (File result : results) {
```

Figure 3.32: iterazione tra i file e inizializzazione livello

Viene analizzato ogni file della directory in un ciclo. **Figure 3.33**

```
File results[] = resultsDir.listFiles();
// Itera attraverso tutti i file nella directory dei risultati della generazione
// di robot da Randoop.

for (File result : results) {
```

Figure 3.33: analisi dei file della directory

Viene calcolata la copertura delle linee di codice del livello corrente estraendolo dal file xml prodotto da Randoop e calcolato il livello corrente prendendolo dal nome del file analizzato. **Figure 3.34** **Figure 3.35** **Figure 3.36**

```
// Calcola la copertura delle linee per ciascun file XML di copertura estraendo  
// il valore dal file XML 'coveragegetot.xml' nella directory corrispondente.  
int score = LineCoverage(result.getAbsolutePath() + "/coveragegetot.xml");
```

Figure 3.34: calcolo copertura

```
// Stampa le informazioni sulla copertura del livello  
System.out.println(  
    result.toString().substring(result.toString().length() - 7, result.toString().length() - 5));
```

Figure 3.35: stampa informazioni

```
// Estra il livello numerico dall'ultimo tratto del nome della directory,  
// basandosi  
// sulla convenzione specifica naming. Nella convenzione attuale, il livello è  
// rappresentato da due caratteri numerici situati nelle posizioni -7 e -5  
// rispetto  
// alla fine del nome della directory  
  
int livello = Integer.parseInt(  
    result.toString().substring(result.toString().length() - 7, result.toString().length() - 5));  
  
System.out.println("La copertura del livello " + String.valueOf(livello) + " è: " + String.valueOf(score));
```

Figure 3.36: estrazione livello

Viene poi creato un oggetto client HTTP per la comunicazione con il task 4. Questo viene impostato per un messaggio con metodo POST per salvare i risultati nel database.

Figure 3.37 **Figure 3.38**

```
// Configurazione di un client HTTP  
HttpClient httpClient = HttpClientBuilder.create().build();
```

Figure 3.37: creazione client

```
// Creazione di un oggetto HttpPost con l'URL "http://t4-g18-app-1:3000/robots"  
HttpPost httpPost = new HttpPost("http://t4-g18-app-1:3000/robots");
```

Figure 3.38: creazione httpPost

Viene creato un array JSON per incapsulare i robot. Questi vengono inizializzati con il punteggio, il tipo, la difficoltà e il nome della classe. **Figure 3.39**

```
// Creazione di un array JSON per contenere le informazioni sui robot generati
JSONArray arr = new JSONArray();

// Creazione di un oggetto JSON per rappresentare un singolo robot generato
JSONObject rob = new JSONObject();

// l'array JSON viene utilizzato per raggruppare gli oggetti JSON che
// rappresentano le informazioni sui robot generati.
// L'array arr contiene una serie di oggetti rob, ognuno dei quali rappresenta
// le caratteristiche di un robot specifico generato da Randoop.

// Aggiunge al robot l'informazione relativa al punteggio convertito in stringa
rob.put("scores", String.valueOf(score));

// aggiunge al robot l'informazione relativa a quale robot è stato utilizzato,
// in questo caso randoop
rob.put("type", "randoop");

// aggiunge al robot l'informazione riguardante il livello di difficoltà
// convertito in stringa
rob.put("difficulty", String.valueOf(livello));

// aggiunge al robot l'informazione relativa all'id della classe di test
rob.put("testClassId", className);

// Aggiunge l'oggetto robot all'array JSON
arr.put(rob);
```

Figure 3.39: creazione array e inizializzazione robot

Viene poi creato un oggetto JSON in cui viene inserito l'array e poi un'entità utilizzando l'oggetto appena creato. **Figure 3.40**

```
// Crea un oggetto JSON principale contenente l'array di robot
JSONObject obj = new JSONObject();

// inserimento dell'array di robot all'interno dell'oggetto
obj.put("robots", arr);

// Crea un'entità JSON utilizzando il contenuto dell'oggetto JSON principale.
StringEntity jsonEntity = new StringEntity(obj.toString(), ContentType.APPLICATION_JSON);
```

Figure 3.40: creazione oggetto Json

Viene configurata la richiesta POST del client, ed eseguita raccogliendo la risposta del server. **Figure 3.41**

```
// Configura la richiesta POST con l'entità JSON creata  
httpPost.setEntity(jsonEntity);  
'
```

Figure 3.41: configurazione richiesta Post

Viene aggiornato il livello corrente con quello del file appena analizzato.

Figure 3.42

```
// esegue la richiesta ed ottiene la risposta  
HttpResponse response = httpClient.execute(httpPost);  
  
// Se il livello del robot generato è superiore al livello massimo attuale,  
// aggiorna il livello massimo.  
if (livello > liv)  
|   liv = livello;  
'
```

Figure 3.42: aggiornamento livello

3.2 1. Documentazione aggiornata

Ricordiamo che il requisito R4 è il seguente: Il Task T1, che consente all'amministratore di caricare una nuova classe da testare, interagisce con T8 e T9 per richiedere la generazione dei test ai 2 Robot. Tali test sono poi salvati nei 1 volumi condivisi T8 e T9. T1 ha successivamente la responsabilità di salvare nel database di T4 i dati di sintesi sulla classe, sui Robot disponibili per essa, e i relativi livelli dei Test disponibili per ogni Robot. Per fare ciò T1 analizza il File system e deduce tali informazioni, per poi salvarle in T4. Verificare la dinamica di questo comportamento di T1 e prevederne una variante (scenario alternativo) che consenta di precaricare nel file system condiviso i Test dei Robot che siano già stati generati e di salvare in T4 i dati di sintesi di questi test.

Il requisito è stato modificato leggermente aggiungendo la possibilità di precaricare non solo i test ma anche la classe relativa in modo da poter giocare subito.

3.2.1 User Story

Come amministratore voglio inserire una nuova classe e i relativi test generati da Evo-Suite e Randoop

- **GIVEN:** L'amministratore ha accesso alla sezione di gestione delle classi ed è già in possesso dei file ZIP contenenti i test.
- **WHEN:** L'amministratore inserisce i dettagli della nuova classe e dei test e clicca sul pulsante di conferma.
- **THEN:** L'elenco delle classi e dei robot da sfidare viene aggiornato.
- **AND:** Gli utenti possono visualizzare e scaricare il codice sorgente della nuova classe.

3.2.2 Use Case Diagram di T1 aggiornato

È stata aggiunta la presenza di un nuovo caso d'uso per la nuova funzionalità: **Figure 3.43**



Figure 3.43: use case diagram aggiornato

3.2.3 Sequence Diagram

Nel sequence diagram aggiornato abbiamo reso anche più esplicite a livello visivo le interazioni con i task T4, T8 e T9. **Figure 3.44**

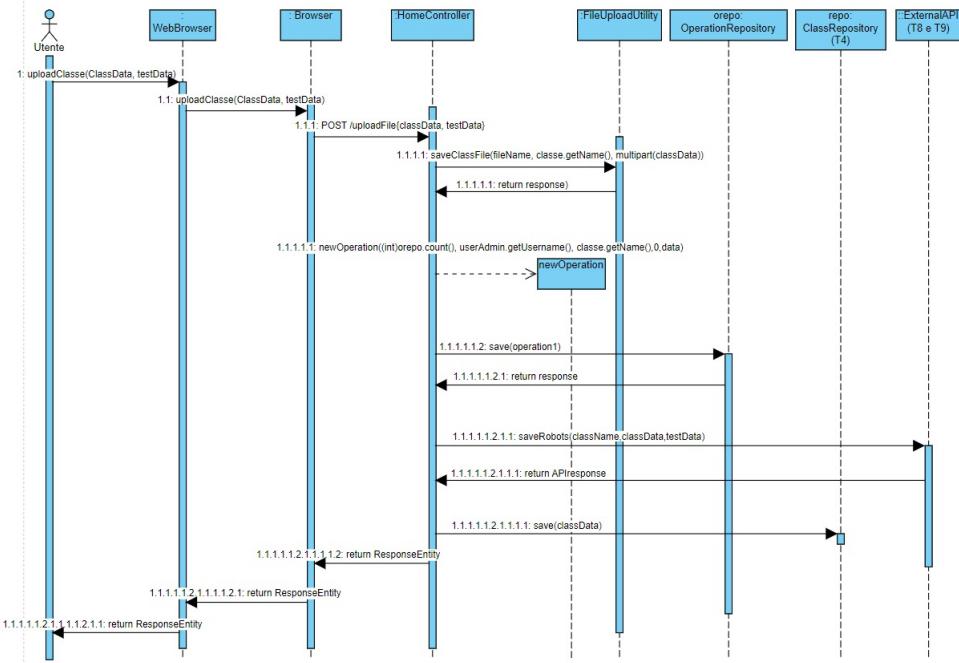


Figure 3.44: sequence diagram aggiornato

3.3 2. Refactoring del metodo generateAndSaveRobots

Prima di passare alla nuova funzionalità è stato eseguito un ulteriore refactoring (rispetto alla seconda iterazione) del codice del precedente scenario di caricamento dei test in modo da mantenere una coerenza tra il vecchio e il nuovo codice. In particolare, per migliorare la leggibilità del codice, la funzione è stata divisa in ulteriori funzioni e ognuna di queste rappresenta una fase specifica del caricamento della classe e la generazione dei test. In generale tutte le funzioni create sono riusate più volte all'interno del codice favorendo il riuso e la manutenibilità dello stesso. Il codice della funzione è il seguente:

Figure 3.45 Figure 3.46 Figure 3.47

```
public static void generateAndSaveRobots(String fileName, String className, MultipartFile classFile) throws IOException {

    // RANDOOP - T9
    Path directory = Paths.get("/VolumeT9/app/FolderTree/" + className + "/" + className + "SourceCode");
    caricaFile(fileName, directory, classFile);

    //Randoop T9
    // creazione del processo esterno di generazione dei test
    ProcessBuilder processBuilder = new ProcessBuilder();

    // con command si configura il comando del processo esterno per eseguire il file
    // JAR 'Task9-G19-0.0.1-SNAPSHOT.jar'
    // l'esecuzione avviene attraverso la JVM di Java.
    // Il parametro "-jar" specifica l'esecuzione di un file JAR.
    processBuilder.command("java", "-jar", "Task9-G19-0.0.1-SNAPSHOT.jar");

    // La directory di lavoro per il processo esterno viene impostata su
    // "/VolumeT9/app/" utilizzando
    // questo metodo garantisce che il processo lavori nella directory desiderata
    processBuilder.directory(new File("/VolumeT9/app"));

    // linea di debug--potremmo anche commentarla
    System.out.println("Prova");

    // si avvia il processo
    Process process = processBuilder.start();

    //Legge l'output del processo appena creato
    outputProcess(process);

    // Crea un oggetto File che rappresenta il percorso della directory contenente i
    // risultati
    // della generazione di robot da Randoop. Il percorso è costruito in base all'ID
    // della classe di test 'className'.
    File resultsDir = new File("/VolumeT9/app/FolderTree/" + className + "/RobotTest/RandoopTest");
}
```

Figure 3.45: codice SaveAndGenerate 1

```
// Inizializza la variabile 'liv' a 0, rappresentante il massimo livello di
// robot prodotti da Randoop.
// Questo valore sarà aggiornato successivamente durante l'analisi dei
// risultati.
int liv = 0; //livelli di robot prodotti da randoop

File results [] = resultsDir.listFiles();

// Itera attraverso tutti i file nella directory dei risultati della generazione
// di robot da Randoop.
for(File result : results) {

    // Calcola la copertura delle linee per ciascun file XML di copertura estraendo
    // il valore dal file XML 'coveragetot.xml' nella directory corrispondente.
    int score = LineCoverage(result.getAbsolutePath() + "/coveragetot.xml");

    System.out.println(result.toString().substring(result.toString().length() - 7, result.toString().length() - 5));

    // Estraie il livello numerico dall'ultimo tratto del nome della directory,
    // basandosi sulla convenzione specifica naming. Nella convenzione attuale:
    // Directory = 0xlivello -> livello = 0x
    int livello = Integer.parseInt(result.toString().substring(result.toString().length() - 7, result.toString().length() - 5));

    System.out.println("La copertura del livello " + String.valueOf(livello) + " è: " + String.valueOf(score));

    saveT4(score, livello, className);

    // Se il livello del robot generato è superiore al livello massimo attuale,
    // aggiorna il livello massimo.
    if(livello > liv)
        liv = livello;
}

}
```

Figure 3.46: codice SaveAndGenerate 2

```
// Il seguente codice è l'adattamento ad evosuite del codice appena visto, i  
// passaggi sono gli stessi  
// EVOSUITE - T8  
// TODO: RICHIENDE AGGIUSTAMENTI IN T8  
Path directoryE = Paths.get("/VolumeT8/FolderTreeEvo/" + className + "/" + className + "SourceCode");  
  
caricaFile(fileName, directoryE, classFile);  
  
ProcessBuilder processBuilderE = new ProcessBuilder();  
  
processBuilderE.command("bash", "robot_generazione.sh", className, "\\\\", "/VolumeT9/app/FolderTree/" + className + "/" +  
className + "SourceCode", String.valueOf(liv));  
processBuilderE.directory(new File("/VolumeT8/Prototipo2.0/"));  
  
Process processE = processBuilderE.start();  
  
outputProcess(processE);  
  
File resultsDirE = new File("/VolumeT8/FolderTreeEvo/" + className + "/RobotTest/EvoSuiteTest");  
  
File resultsE [] = resultsDirE.listFiles();  
for(File result : resultsE) {  
    int score = LineCoverageE(result.getAbsolutePath() + "/TestReport/statistics.csv");  
  
    System.out.println(result.toString().substring(result.toString().length() - 7, result.toString().length() - 5));  
    int livello = Integer.parseInt(result.toString().substring(result.toString().length() - 7, result.toString().length() -  
5));  
  
    System.out.println("La copertura del livello " + String.valueOf(livello) + " è: " + String.valueOf(score));  
  
    saveT4(score, livello, className);  
}  
}
```

Figure 3.47: codice SaveAndGenerate 3

Adesso vediamo le varie fasi nel dettaglio.

3.3.1 A. Caricamento della classe nel Filesystem

Questa fase è racchiusa nella funzione caricaFile. **Figure 3.48**

```
public static void caricaFile(String fileName, Path directory, MultipartFile file) throws IOException{
    try {
        // Verifica se la directory esiste già
        if (!Files.exists(directory)) {
            // Crea la directory
            Files.createDirectories(directory);
            System.out.println("La directory è stata creata con successo.");
        } else {
            System.out.println("La directory esiste già.");
        }
    } catch (Exception e) {
        System.out.println("Errore durante la creazione della directory: " + e.getMessage());
    }
    // Legge l'input stream del file caricato e lo copia nella directory specificata
    try (InputStream inputStream = file.getInputStream()) {
        // Risolve il percorso completo del file all'interno della directory
        // specificata.
        // Viene utilizzato il metodo 'directory.resolve(fileNameClass)' per ottenere il
        // percorso completo
        // del file all'interno della directory 'directory'. Questo percorso completo
        // sarà utilizzato
        // successivamente per copiare il contenuto dell'input stream del file nella
        // posizione desiderata.
        Path filePath = directory.resolve(fileName);
        System.out.println(filePath.toString());

        // copio il contenuto dell'input stream nel file di destinazione
        // l'ultimo parametro di questa funzione indica che se il file già esiste deve
        // essere sostituito
        Files.copy(inputStream, filePath, StandardCopyOption.REPLACE_EXISTING);

        // chiusura dell'input stream dopo aver completato la copia
        inputStream.close();
    }
}
```

Figure 3.48: codice caricaFile

3.3.2 B. Generazione dei test: lettura dell'output del processo

Questa fase è racchiusa nella funzione outputProcess. Legge e stampa sulla console di sistema.

Figure 3.49

```
public static void outputProcess(Process process) throws IOException{
    // Legge l'output del processo esterno tramite un BufferedReader, che a sua
    // volta usa
    // un InputStreamReader per convertire i byte in caratteri. Il metodo
    // 'process.getInputStream()'
    // restituisce lo stream di input del processo esterno.
    BufferedReader reader = new BufferedReader(new InputStreamReader(process.getInputStream()));
    String line;

    // All'interno del loop viene letta ogni linea disponibile finché il processo
    // continua a produrre output.
    while ((line = reader.readLine()) != null)
        System.out.println(line);

    // funzionamento analogo al precedente, invece di leggere l'output leggiamo gli
    // errori
    reader = new BufferedReader(new InputStreamReader(process.getErrorStream()));
    while ((line = reader.readLine()) != null)
        System.out.println(line);

    try {
        // Attende che il processo termini e restituisce il codice di uscita
        int exitCode = process.waitFor();

        System.out.println("ERRORE CODE: " + exitCode);
    } catch (InterruptedException e) {
        System.out.println(e);
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```

Figure 3.49: codice outputProcess

3.3.3 C. Salvataggio dei dati nel Task T4

Questa fase è racchiusa nella funzione **saveT4**. Comunica con il task T4 per salvare tutte le informazioni relative ai test condotti dai Robot a valle della valutazione dei risultati.

Figure 3.45 Figure 3.46

```
public static void saveT4(int score, int livello, String className, String robotName) throws IOException{
    // Configurazione di un client HTTP
    HttpClient httpClient = HttpClientBuilder.create().build();

    // Creazione di un oggetto HttpPost con l'URL "http://t4-g18-app-1:3000/robots"
    HttpPost httpPost = new HttpPost("http://t4-g18-app-1:3000/robots");

    // Creazione di un array JSON per contenere le informazioni sui robot generati
    JSONArray arr = new JSONArray();

    // Creazione di un oggetto JSON per rappresentare un singolo robot generato
    JSONObject rob = new JSONObject();

    // L'array JSON viene utilizzato per raggruppare gli oggetti JSON che
    // rappresentano le informazioni sui robot generati.
    // L'array arr contiene una serie di oggetti rob, ognuno dei quali rappresenta

    // Aggiunge al robot l'informazione relativa al punteggio convertito in stringa
    rob.put("scores", String.valueOf(score));

    // aggiunge al robot l'informazione relativa a quale robot è stato utilizzato,
    rob.put("type", robotName);

    // aggiunge al robot l'informazione riguardante il livello di difficoltà
    // convertito in stringa
    rob.put("difficulty", String.valueOf(livello));

    // aggiunge al robot l'informazione relativa all'id della classe di test
    rob.put("testClassId", className);

    // Aggiunge l'oggetto robot all'array JSON
    arr.put(rob);

    // Crea un oggetto JSON principale contenente l'array di robot
    JSONObject obj = new JSONObject();
```

Figure 3.50: codice saveT4 1

```
// inserimento dell'array di robot all'interno dell'oggetto  
obj.put("robots", arr);  
  
// Crea un'entità JSON utilizzando il contenuto dell'oggetto JSON principale.  
StringEntity jsonEntity = new StringEntity(obj.toString(), ContentType.APPLICATION_JSON);  
  
// Configura la richiesta POST con l'entità JSON creata  
httpPost.setEntity(jsonEntity);  
  
// esegue la richiesta ed ottiene la risposta  
HttpResponse response = httpClient.execute(httpPost);  
}
```

Figure 3.51: codice saveT4 2

3.4 3. Struttura dei file prodotti dai robot:

Prima di passare alla nuova funzionalità è stato necessario capire sotto quale forma caricare tutti i file dei test e che struttura hanno i risultati della generazione. Questa struttura deve essere rispettata dai file zip caricati per garantire un corretto funzionamento dell'applicazione. Ne assumiamo la correttezza in quanto di default i robot generano i test restituendo le strutture che vedremo ora. La struttura per Randoop è la seguente:

```
+01level  
  -coveragegetot.xml  
+02level  
  -coveragegetot.xml  
+03level  
  -coveragegetot.xml  
|  
|  
|  
+xxlevel  
  -coveragegetot.xml
```

La struttura per Evosuite è la seguente:

```
+01level
  +TestReport
    -statistics.csv
+02level
  +TestReport
    -statistics.csv
+03level
  -+TestReport
    -statistics.csv
|
|
|
+xxlevel
  +TestReport
    -statistics.csv
```

Per rendere semplice e veloce l'inserimento dei file dal front-end, abbiamo deciso di rendere possibile il caricamento di questa struttura all'interno di un file .zip che verrà successivamente scompattato dal back-end. Quest'ultimo, inoltre, eseguirà anche il rename dell'archivio in modo da rendere il file trattabile dalla funzione che estrae i file dall'archivio. La convenzione decisa per il nome dei file zip carichi è: **|nome-Classe|Test|nomeRobot|.zip**. Nel caso il nome non segua la convenzione decisa il caricamento avverrà lo stesso e il nome verrà modificato secondo la convenzione. Ad esempio, io come amministratore voglio caricare un file zip relativo a test della classe "Range". Il mio file si chiama "Sole1.zip", il caricamento avverrà lo stesso e il file verrà rinominato in "RangeTestRandoop.zip" o in "RangeTestEvosuite.zip" in base al robot.

Inoltre, durante lo sviluppo della nuova funzione, ci siamo accorti che non era possibile caricare file superiori a una certa dimensione imposta da Spring di default. Abbiamo quindi modificato i file **default.conf** presenti in **ui gateway** aggiungendo:

```
client_max_body_size 5M;  
client_body_timeout 60s;
```

e **application.properties**:

```
spring.servlet.multipart.max-file-size=5MB  
spring.servlet.multipart.max-request-size=5MB
```

3.5 4. Modifiche alla pagina home_adm:

È stato aggiunto un nuovo elemento nella barra di navigazione della home per gli admin in modo da non creare confusione nel form già esistente per il caricamento e la generazione di test. **Figure 3.52**

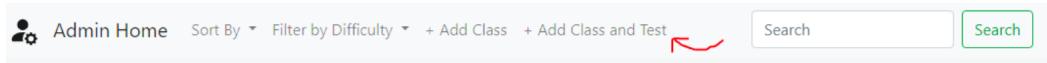


Figure 3.52: modifica barra di navigazione

Il codice aggiunto ad `home_adm.html` è il seguente:

Figure 3.53

```
<li class="nav-item">  
  <a class="nav-link" href="/uploadClasseAndTest">+ Add Class and Test</a>  
</li>
```

Figure 3.53: codice per modifica barra di navigazione

3.6 5. Pagina uploadClasseAndTest:

La nuova pagina appare molto simile alla precedente con l'aggiunta dell'input del file zip per i test.

Figure 3.54

Test upload

Class name

Date

gg/mm/aaaa

Difficulty: Beginner

Select the difficulty level of this item

Description

Category 1

Category 2

Category 3

Upload your class (.java): Nessun file scelto

Upload your test Randoop (.zip): Nessun file scelto

Upload your test Evosuite (.zip): Nessun file scelto

Upload class and tests

Figure 3.54: nuova pagina

Il codice aggiunto rispetto alla pagina già esistente è il seguente: **Figure 3.55 Figure 3.56**

```

<div class="form-group mt-3">
  <label class="mr-2">Upload your test Randoop (.zip) :</label>
  <input type="file" name="test" id="testInput">
</div>

<div class="form-group mt-3">
  <label class="mr-2">Upload your test Evosuite (.zip) :</label>
  <input type="file" name="testEvo" id="testInputEvo">
</div>

<div class="form-group mt-3">
  <button type="button" onclick="uploadTest()" class="btn btn-success">Upload class and tests</button>
</div>

```

Figure 3.55: codice aggiunto alla pagina 1

```

<script>
    //si aggancia al form tramite ID
    const formTest = document.getElementById('formId');

    //Funzione per l'upload della classe e dei relativi test
    function uploadTest(event) {
        //Creazione delle costanti che rappresentano i dati input al form
        const name = document.getElementById('className').value;
        const date = document.getElementById('date').value;
        const difficulty = document.getElementById('difficulty').value;
        const code_Url = "";
        const description = document.getElementById('description').value;
        const category = [
            document.getElementById('category1').value,
            document.getElementById('category2').value,
            document.getElementById('category3').value
        ];

        //Costante che rappresenta il file .java in input della classe
        const classInput = document.getElementById('fileInput');
        const file = classInput.files[0];
        //Costante che rappresenta il file .zip in input dei test Randoop
        const testInput = document.getElementById('testInput');
        const test = testInput.files[0];
        //Costante che rappresenta il file .zip in input dei test Randoop
        const testInputEvo = document.getElementById('testInputEvo');
        const testEvo = testInputEvo.files[0];

        //Costante che rappresenta l'insieme dei dati del form e i file in input
        const formData = new FormData();
        formData.append('file', file);
        formData.append('model', JSON.stringify({
            name: name,
            date: date,
            difficulty: difficulty,
            code_Url: code_Url,
            description: description,
            category: category
        }));
        formData.append('test', test);
        formData.append('testEvo', testEvo);

        //chiamata HTTP alla funzione uploadTest in HomeController con metodo POST e dati del form come body
        fetch('/uploadTest', {
            method: 'POST',
            body: formData
        })
        .then(response => response.json())
        .then(data => {
            console.log("Success:", data);
            window.location.href = "/home_adm";
            // Aggiungi qui il codice per gestire la risposta dal server
        })
        .catch(error => {
            console.error('Error:', error);
            // Aggiungi qui il codice per gestire gli errori
        });
    }
</script>

```

Figure 3.56: codice aggiunto alla pagina 2

3.7 6. Modifiche al controller:

Per mappare la nuova pagina e collegarla al controller sono state fatte le seguenti modifiche:

3.7.1 default.conf:

```
13. location ~  
^/(loginAdmin|registraAdmin|home_adm|modificaClasse|orderbydate|  
Dfilterby.+|orderbyname|Reports|uploadClasse|uploadClasseAndTest|  
reportClasse|delete|getLikes|uploadFile|uploadTest|home|t1) {  
14.     include /etc/nginx/includes/proxy.conf;  
15.     proxy_pass http://manvsclass-controller-1:8080;  
16. }
```

3.7.2 HomeController.java:

Figure 3.57

```
@GetMapping("/uploadClasseAndTest")  
public String showUploadClasseAndTest() {  
    return "uploadClasseAndTest";  
}
```

Figure 3.57: mapping nuova rottta

Utilizza l'annotazione @GetMapping per mappare una richiesta HTTP GET all' endpoint /uploadClasseAndTest. Inoltre, è stato aggiunto un nuovo metodo per la richiesta HTTP POST di caricamento dei file della classe e dei test: **Figure 3.58**

CHAPTER 3. REQUISITO R4

```
@PostMapping("/uploadTest")
@ResponseEntity<FileUploadResponse>
public ResponseEntity<FileUploadResponse> uploadTest(@RequestParam("file") MultipartFile classFile, @RequestParam("model") String model, @RequestParam("test") MultipartFile testFile,
                                                       @RequestParam("testEvo") MultipartFile testFileEvo) throws IOException {
    //Legge i metadati della classe della parte "model" del body HTTP e li salva in un oggetto ClasseUf
    ObjectMapper mapper = new ObjectMapper();
    ClasseUf classe = mapper.readValue(model, ClasseUf.class);

    //Salvo il nome del file della classe caricato
    String fileNameClass = StringUtils.cleanPath(classFile.getOriginalFilename());
    long size = classFile.getSize();

    //Salvo la classe nel filesystem condiviso
    FileUploadUtil.saveClassFile(fileNameClass, classe.getName(), classFile);

    //Salvo i test nel filesystem condiviso
    String fileNameTest = StringUtils.cleanPath(testFile.getOriginalFilename());
    String fileNameTestEvo = StringUtils.cleanPath(testFileEvo.getOriginalFilename());
    RobotUtil.saveRobots(fileNameClass, fileNameTest, fileNameTestEvo, classe.getName(), classFile, testFile, testFileEvo);

    FileUploadResponse response = new FileUploadResponse();
    response.setFileName(fileNameClass);
    response.setSize(size);
    response.setDownloadUrl("/downloadFile");

    //Setta data di caricamento e percorso di download della classe
    classe.setUrl("files-upload/" + classe.getName() + "/" + fileNameClass);
    classe.setDate(today.toString());

    //Creazione dell'oggetto riguardante l'operazione appena fatta
    LocalDate currentDate = LocalDate.now();
    DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd");
    String data = currentDate.format(formatter);
    Operation operation = new Operation((int) repo.count(), userAdmin.getUsername(), classe.getName() + " con Robot", 0, data);

    //Salvo i dati sull'operazione fatta nel database
    operation.save(operation);
    //Salvo i dati sulla classe nel database
    repo.save(classe);
    return new ResponseEntity<>(response, HttpStatus.OK);
}
```

Figure 3.58: codice per la richiesta POST

3.8 7. Metodo saveRobots:

Questo metodo fa parte della classe **RobotUtil** situata nel package del **filesystem** all'interno del **model** e interagisce con il **T4**. Il diagramma di attività del metodo **saveRobots** è il seguente: **Figure 3.58**

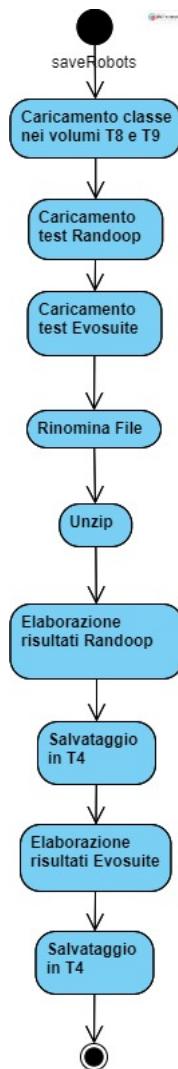


Figure 3.59: diagramma attività saveRobots

Sfruttando il refactoring del metodo `generateAndSaveRobots` della classe `RobotUtil`, è stato scritto un nuovo metodo per soddisfare il requisito R4. Il file .zip viene caricato

all'interno del filesystem condiviso del robot e subito dopo viene rinominato il file in modo da uniformarsi alla convenzione scelta (3), vengono estratte le cartelle con la struttura vista (3) e infine viene cancellato l'archivio. Il resto dell'elaborazione rimane pressoché identica. Di seguito vediamo il codice:

a. CARICAMENTO CLASSE E TEST + CONTROLLO SUL NOME DELLE ZIP: Figure 3.60

```
/*CARICAMENTO CLASSE NEI VOLUMI T8 E T9*/
Path directory = Paths.get("/VolumeT9/app/FolderTree/" + className + "/" + className + "SourceCode");
Path directoryEvo = Paths.get("/VolumeT8/FolderTreeEvo/" + className + "/" + className + "SourceCode");
caricaFile(fileNameClass, directory, classFile);
caricaFile(fileNameClass, directoryEvo, classFile);

/*CARICAMENTO TEST */
//Randoop
Path directoryTest = Paths.get("/VolumeT9/app/FolderTree/" + className + "/RobotTest/RandoopTest");
caricaFile(fileNameTest, directoryTest, testFile);
//Evosuite
Path directoryTestEvo = Paths.get("/VolumeT8/FolderTreeEvo/" + className + "/RobotTest/EvoSuiteTest");
caricaFile(fileNameTestEvo, directoryTestEvo, testFileEvo);

//Rinomina il file zip caricato secondo la convenzione attuale: |nomeClasse|TestRandoop.zip
File fileZipDir = new File("/VolumeT9/app/FolderTree/" + className + "/RobotTest/RandoopTest/");
File fileZip[] = fileZipDir.listFiles();
String nomeAttuale = fileZip[0].getAbsolutePath().toString();
String nuovoNome = "/VolumeT9/app/FolderTree/" + className + "/RobotTest/RandoopTest/" + className + "TestRandoop.zip";
File zipAttuale = new File(nomeAttuale);
File zipNuova = new File(nuovoNome);
boolean rinominato = zipAttuale.renameTo(zipNuova);

//Rinomina il file zip caricato secondo la convenzione attuale: |nomeClasse|TestEvosuite.zip
File fileZipDirEvo = new File("/VolumeT8/FolderTreeEvo/" + className + "/RobotTest/EvoSuiteTest");
File fileZipEvo[] = fileZipDirEvo.listFiles();
String nomeAttualeEvo = fileZipEvo[0].getAbsolutePath().toString();
String nuovoNomeEvo = "/VolumeT8/FolderTreeEvo/" + className + "/RobotTest/EvoSuiteTest/" + className + "TestEvosuite.zip";
File zipAttualeEvo = new File(nomeAttualeEvo);
File zipNuovaEvo = new File(nuovoNomeEvo);
boolean rinominatoEvo = zipAttualeEvo.renameTo(zipNuovaEvo);
```

Figure 3.60: codice per il carimento di classe e test

b. UNZIP **Figure 3.61**

```
//----- FUNZIONE UNZIP -----
public static File newFile(File destinationDir, ZipEntry zipEntry) throws IOException {
    File destfile = new File(destinationDir, zipEntry.getName());
    String destDirPath = destinationDir.getCanonicalPath();
    String destFilePath = destfile.getCanonicalPath();

    if (!destFilePath.startsWith(destDirPath + File.separator)) {
        throw new IOException("Entry is outside of the target dir: " + zipEntry.getName());
    }
    return destfile;
}

public static void unzip(String className) throws IOException {
    String fileZip = "/VolumeT9/app/FolderTree/" + className + "/RobotTest/RandoopTest/" + className + "TestRandoop.zip";
    File destDir = new File("/VolumeT9/app/FolderTree/" + className + "/RobotTest/RandoopTest/");

    byte[] buffer = new byte[1024];
    ZipInputStream zis = new ZipInputStream(new FileInputStream(fileZip));
    ZipEntry zipEntry = zis.getNextEntry();
    while (zipEntry != null) {
        File newFile = newFile(destDir, zipEntry);
        if (zipEntry.isDirectory()) {
            if (!newFile.mkdirs()) {
                throw new IOException("Failed to create directory " + newFile);
            }
        } else {
            // fix for Windows-created archives
            File parent = newFile.getParentFile();
            if (!parent.isDirectory() && !parent.mkdirs()) {
                throw new IOException("Failed to create directory " + parent);
            }

            // write file content
            FileOutputStream fos = new FileOutputStream(newFile);
            int len;
            while ((len = zis.read(buffer)) > 0) {
                fos.write(buffer, 0, len);
            }
            fos.close();
        }
        zipEntry = zis.getNextEntry();
    }
    zis.closeEntry();
    zis.close();
}
```

Figure 3.61: codice per unzip

c. ELABORAZIONE RISULTATI RANDOOP: Figure 3.62

```

/*SALVATAGGIO RISULTATI NEL TASK T4 */

// Crea un oggetto File che rappresenta il percorso della directory contenente i
// risultati
// della generazione di robot da Randoop. Il percorso è costruito in base all'ID
// della classe di test 'className'.
File resultsDir = new File("/VolumeT9/app/FolderTree/" + className + "/RobotTest/RandoopTest");

// Inizializza la variabile 'liv' a 0, rappresentante il massimo livello di
// robot prodotti da Randoop.
// Questo valore sarà aggiornato successivamente durante l'analisi dei
// risultati.
int liv = 0; // livelli di robot prodotti da randoop

File results[] = resultsDir.listFiles();

// Itera attraverso tutti i file nella directory dei risultati della generazione
// di robot da Randoop.

for (File result : results) {

    // Calcola la copertura delle linee per ciascun file XML di copertura estraendo
    // il valore dal file XML 'coveragetot.xml' nella directory corrispondente.
    int score = LineCoverage(result.getAbsolutePath() + "/coveragetot.xml");
    // Stampa le informazioni sulla copertura del livello
    System.out.println(
        result.toString().substring(result.toString().length() - 7, result.toString().length() - 5));

    // Estraie il livello numerico dall'ultimo tratto del nome della directory,
    // basandosi sulla convenzione specifica naming. Nella convenzione attuale:
    // Directory = 0xlivello -> livello = 0x
    int livello = Integer.parseInt(
        result.toString().substring(result.toString().length() - 7, result.toString().length() - 5));

    System.out.println("La copertura del livello " + String.valueOf(livello) + ":" + String.valueOf(score));

    saveT4(score, livello, className);

    // Se il livello del robot generato è superiore al livello massimo attuale,
    // aggiorna il livello massimo.
    if (livello > liv)
        liv = livello;
}

```

Figure 3.62: codice per risultati randoop

d. ELABORAZIONE RISULTATI EVOSUITE: Figure 3.63

```

/*TODO: AGGIUSTAMENTI T8 PER EVOSUITE */
// Il seguente codice è l'adattamento ad evosuite del codice appena visto,
// passaggi sono gli stessi
File resultsDirEvo = new File("/VolumeT8/FolderTreeEvo/" + className + "/RobotTest/EvoSuiteTest");

File resultsEvo [] = resultsDirEvo.listFiles();
for(File result : resultsEvo) {
    int score = LineCoverage(result.getAbsolutePath() + "/TestReport/statistics.csv");

    System.out.println(result.toString().substring(result.toString().length() - 7, result.toString().length() - 5));
    int livello = Integer.parseInt(result.toString().substring(result.toString().length() - 7, result.toString().length() - 5));

    System.out.println("La copertura del livello " + String.valueOf(livello) + ":" + String.valueOf(score));

    saveT4(score, livello, className);
}

```

Figure 3.63: codice per risultati evosuite

La funzione di unzip ha il seguente **diagramma di attività**: **Figure 3.64**

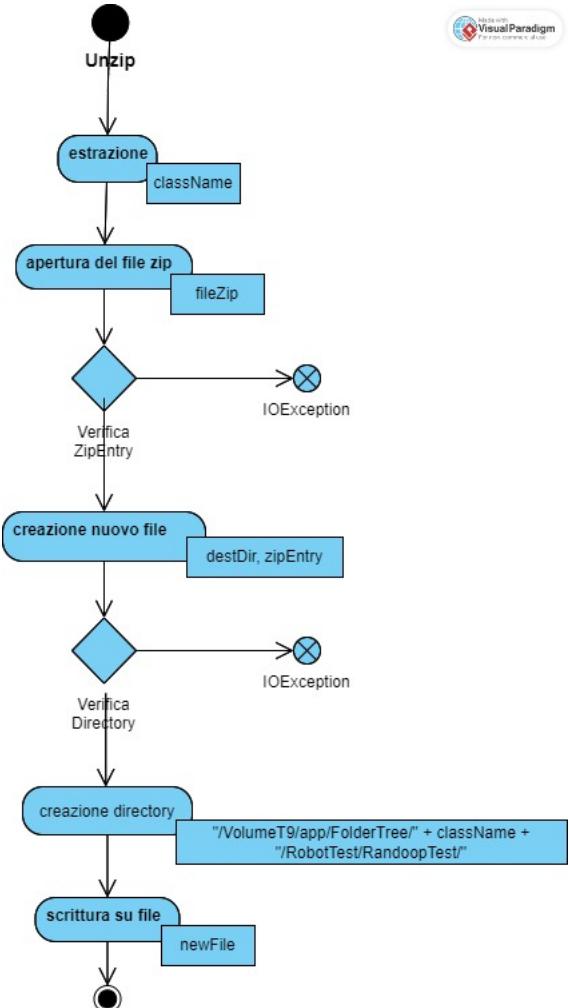


Figure 3.64: diagramma di attività unzip

Per realizzare la funzione abbiamo fatto uso della libreria java.util.zip:

Figure 3.65

```
-----FUNZIONE UNZIP
public static File newFile(File destinationDir, ZipEntry zipEntry) throws IOException {
    File destFile = new File(destinationDir, zipEntry.getName());
    String destDirPath = destinationDir.getCanonicalPath();
    String destFilePath = destFile.getCanonicalPath();

    if (!destFilePath.startsWith(destDirPath + File.separator)) {
        throw new IOException("Entry is outside of the target dir: " + zipEntry.getName());
    }

    return destFile;
}
public static void unzip(String className) throws IOException {
    String fileZip = "/VolumeT9/app/FolderTree/" + className + "/RobotTest/RandoopTest/" + className + "TestRandoop.zip";
    File destDir = new File("/VolumeT9/app/FolderTree/" + className + "/RobotTest/RandoopTest/");

    byte[] buffer = new byte[1024];
    ZipInputStream zis = new ZipInputStream(new FileInputStream(fileZip));
    ZipEntry zipEntry = zis.getNextEntry();
    while (zipEntry != null) {
        File newFile = newFile(destDir, zipEntry);
        if (zipEntry.isDirectory()) {
            if (!newFile.isDirectory() && !newFile.mkdirs()) {
                throw new IOException("Failed to create directory " + newFile);
            }
        } else {
            // fix for Windows-created archives
            File parent = newFile.getParentFile();
            if (!parent.isDirectory() && !parent.mkdirs()) {
                throw new IOException("Failed to create directory " + parent);
            }

            // write file content
            FileOutputStream fos = new FileOutputStream(newFile);
            int len;
            while ((len = zis.read(buffer)) > 0) {
                fos.write(buffer, 0, len);
            }
            fos.close();
        }
        zipEntry = zis.getNextEntry();
    }
    zis.closeEntry();
    zis.close();
}
```

Figure 3.65: codice unzip con libreria

3.9 Testing

Per testare il funzionamento abbiamo semplicemente simulato il flusso di esecuzione normale del requisito implementato. Abbiamo provato ad utilizzare il programma nel ruolo di amministratore e abbiamo caricato un file zip contenente i test. In particolare abbiamo simulato lo scenario in cui il nome del file zip non sia conforme alla struttura che abbiamo specificato in precedenza. Non effettuiamo un controllo sul contenuto del file, assumiamo che la sua struttura sia corretta essendo stato generato da un robot.

L'unico controllo che viene effettuato è sul file zip, in caso di nome non conforme alla convenzione scelta, il funzionamento rimane inalterato, si corregge automaticamente il nome del file. Dopo il caricamento il test appare immediatamente nel gioco. Ricordiamo che è presente un video allegato nella repository di github. In questo video mostriamo passo passo ciò che farebbe un amministratore quando accede al gioco e desidera caricare una classe di test.

3.10 Considerazioni finali

L'implementazione del requisito R4 non modifica l'architettura dell'applicazione in maniera evidente. Il caricamento della classe di test direttamente dal file system dell'amministratore elimina la dipendenza dai task T8 e T9 in quanto non è più necessaria la generazione dei test da parte dei robot EvoSuite e Randoop. Rispetto alla versione originale, il numero di righe di codice non è aumentato particolarmente, sono aumentate le funzioni in quanto avendo usato il pattern extract method da un metodo originale ne abbiamo generati diversi.

Chapter 4

Requisito R6

4.1 User story

Il task R6 richiedeva l'implementazione di una versione in lingua inglese dell'applicazione.

- **GIVEN:** L'utente è nella schermata principale del gioco.
- **WHEN:** L'utente sceglie di utilizzare il gioco in lingua inglese.
- **THEN:** Tutti i testi, dialoghi e informazioni del gioco sono presentati in inglese per arricchire l'esperienza linguistica dell'utente.
- **AND:** Le pagine HTML associate al gioco sono automaticamente tradotte in inglese, consentendo all'utente di accedere alle risorse online nella nuova lingua.

Lo approccio inizialmente scelto è stato quello di modificare le rotte esistenti che restituivano pagine HTML, premettendo a tale percorso un ulteriore livello, che identifica la lingua. Ad esempio:

<https://localhost:80/it/modificaClasse> per la pagina in italiano;

<https://localhost:80/en/modificaClasse> per la pagina in inglese.

A tale struttura dovrà chiaramente corrispondere una gerarchia analoga della directory in cui le pagine HTML sono contenute, distinguendo una cartella `it` per le pagine in italiano ed una cartella `en` per le pagine in inglese.

L'approccio si rivela in realtà fortemente scalabile, in quanto al netto delle modifiche effettuate in questa iterazione e di una minima aggiunta al file di configurazione del reverse proxy, permette di aggiungere altre cartelle contenenti le pagine HTML tradotte in qualsiasi altra lingua (ad esempio, `es` per lo spagnolo, `fr` per il francese).

4.1.1 Installazione di Maven

Lavorando con l'IDE Visual Studio Code, è stato necessario installare Maven, ovvero uno strumento di gestione di progetti software ampiamente utilizzato nell'ecosistema Java per automatizzare il processo di build di un progetto, la gestione delle dipendenze, la distribuzione e la documentazione del software.

- 1. Download del Binary zip archive. **Maven Download Apache Maven** Apache Maven 3.9.6 is the latest release: it is the recommended version for all users.
<https://maven.apache.org/download.cgi>
- 2. Aggiunta del percorso dell'archivio alla variabile d'ambiente **PATH**.
- 3. Posizionarsi con la shell nella directory `projectpath/T1-G11/applicazione/manvsclass.4.Eseguire`
- 5. **Rieseguire lo script installer.bat.**

Path: `T1-G11/applicazione/manvsclass/src/main/resources/templates`

```
templates \
|
| --> it \
|   --> home.html
```

```
| --> home_adm.html  
| --> login_admin.html  
| --> modificaClasse.html  
| --> registraAdmin.html  
| --> reportClasse.html  
| --> Reports.html  
| --> uploadClasse.html  
  
| --> en \  
| --> home.html  
| --> home_adm.html  
| --> login_admin.html  
| --> modificaClasse.html  
| --> registraAdmin.html  
| --> reportClasse.html  
| --> Reports.html  
| --> uploadClasse.html
```

Path: T5-G2/t5/src/main/resources/templates

```
templates \  
|  
| --> it \  
| --> change_password.html  
| --> editor.html  
| --> login.html  
| --> main.html  
| --> report.html  
  
| --> en \  
| --> change_password.html  
| --> editor.html
```

```
| --> login.html  
| --> main.html  
| --> report.html  
  
Path: T23-G1/src/main/resources/templates  
  
templates \  
|  
| --> it \  
|   | --> login.html  
|   | --> mail_register.html  
|   | --> password_change.html  
|   | --> password_reset.html  
|   | --> register.html  
| --> en \  
|   | --> login.html  
|   | --> mail_register.html  
|   | --> password_change.html  
|   | --> password_reset.html  
|   | --> register.html
```

4.2 HomeController.java (T1)

Il framework Spring, il quale implementa il pattern Model-View-Controller (MVC) permette di definire all'interno del Controller delle rotte che restituiscono pagine HTML dinamicamente (Views). La sintassi è simile alla seguente:

```
@Controller  
public class WelcomeController {
```

```
@GetMapping("/welcome")
public String showWelcomePage() {
    // Ritorna il nome del file HTML (senza estensione) da
    // visualizzare
    return "welcome";
}
```

Nel nostro caso, è stato necessario anteporre un parametro (chiamato coerentemente language) che viene letto come variabile e utilizzato per costruire il percorso del file HTML all'interno della directory, divisa ora in una cartella it/ e una cartella en/.

```
Path: T1-G11/applicazione/manvsclass/src/main/java/com/groom/
manvsclass/controller

@GetMapping("{language}/home_adm")
public String showHomeAdmin(@PathVariable String language) {
    return language + "/home_adm";
}

@GetMapping("{language}/loginAdmin")
public String showLoginAdmin(@PathVariable String language) {
    return language + "/login_admin";
}

@GetMapping("{language}/registraAdmin")
public String showRegistraAdmin(@PathVariable String language) {
    return language + "/registraAdmin";
}

@GetMapping("{language}/modificaClasse")
public String showModificaClasse(@PathVariable String language) {
    return language + "/modificaClasse";
```

```

}

@GetMapping("/{language}/uploadClasse")
public String showUploadClasse(@PathVariable String language) {
    return language + "/uploadClasse";
}

@GetMapping("/{language}/reportClasse")
public String showReportClasse(@PathVariable String language) {
    return language + "/reportClasse";
}

@GetMapping("/{language}/Reports")
public String showReports(@PathVariable String language) {
    return language + "/Reports";
}

```

Path: T5-G2/t5/src/main/java/com/g2/t5

```

@GetMapping("/{language}/main")
public String GUIController( @PathVariable String language,
Model model, @CookieValue(name ="jwt", required = false) String jwt){
    MultiValueMap<String, String> formData = new LinkedMultiValueMap<
        String, String>();
    formData.add("jwt", jwt);
    Boolean isAuthenticated = restTemplate.postForObject
        ("http://t23-g1-app-1:8080/validateToken", formData, Boolean.class
        );
    if(isAuthenticated == null || !isAuthenticated)
        return "redirect:/" + language + "/login";
    // ...
    return langauge + "/main";
}

```

```
// -----
@GetMapping("{language}/report")
public String reportPage( @PathVariable String language,
Model model, @CookieValue(name = "jwt", required = false) String jwt)
{
    MultiValueMap<String, String> formData = new LinkedMultiValueMap<
        String, String>();
    formData.add("jwt", jwt);
    Boolean isAuthenticated = restTemplate.postForObject
        ("http://t23-g1-app-1:8080/validateToken", formData, Boolean.class
    );
    if(isAuthenticated == null || !isAuthenticated)
        return "redirect:/" + language + "/login";
    return language + "/report";
}
// -----
@GetMapping("{language}/editor")
public String editorPage( @PathVariable String language,
Model model, @CookieValue(name = "jwt", required = false) String jwt)
{
    MultiValueMap<String, String> formData = new LinkedMultiValueMap<
        String, String>();
    formData.add("jwt", jwt);
    Boolean isAuthenticated = restTemplate.postForObject
        ("http://t23-g1-app-1:8080/validateToken", formData, Boolean.class
    );
    if(isAuthenticated == null || !isAuthenticated)
        return "redirect:/" + language + "/login";
    return language + "/editor";
}
```

```
Path: T23-G1/src/main/java/com/example/db_setup

@GetMapping("/{language}/register")
public ModelAndView showRegistrationForm
(@PathVariable String language,
HttpServletRequest request, @CookieValue(name = "jwt", required =
false) String jwt) {
    if(isJwtValid(jwt))
        return new ModelAndView("redirect:http://localhost/" + language
        + "/main");
    return new ModelAndView(language + "/register");
}

// -----
@GetMapping("/{language}/login")
public ModelAndView showLoginForm
(@PathVariable String language,
HttpServletRequest request, @CookieValue(name="jwt", required=false)
String jwt) {
    if(isJwtValid(jwt))
        return new ModelAndView("redirect:http://localhost/" + language
        + "/main");
    return new ModelAndView(language + "/login");
}

// -----
@GetMapping("/{language}/password_reset")
public ModelAndView showResetForm (@PathVariable String language,
HttpServletRequest request, @CookieValue(name="jwt", required=
false) String jwt) {
    if(isJwtValid(jwt))
        return new ModelAndView("redirect:http://localhost/" + language
```

```
+ "/main");

return new ModelAndView(language + "/password_reset");
}

// -----
@GetMapping("/{language}/password_change")
public ModelAndView showChangeFor (@PathVariable String language,
HttpServletRequest request, @CookieValue(name="jwt", required=false)
String jwt) {
if(isJwtValid(jwt))

    return new ModelAndView("redirect:http://localhost/" + language
+ "/main");

return new ModelAndView(language + "/password_change");
}

@GetMapping("/{language}/mail_register")
public ModelAndView showMailForm (@PathVariable String language,
HttpServletRequest request, @CookieValue(name="jwt", required=false)
String jwt) {
if(isJwtValid(jwt))

    return new ModelAndView("redirect:http://localhost/" + language
+ "/main");

return new ModelAndView(language + "/mail_register");
}
```

4.3 Controller.java (T23) e GuiController.java (T5)

Modifiche analoghe sono state apportate anche ai controller dei componenti T5 e T23. E' stata aggiunta la lingua ad ogni metodo relativo alle richieste Get, mentre le richieste

post sono state lasciate inalterate.

4.4 Altre risorse modificate

Il refactoring ha richiesto un aggiornamento di tutti i file, sia frontend (HTML e JavaScript) che backend (le stesse classi Java in cui avviene un reindirizzamento) i quali avevano un riferimento ad una delle rotte modificate, ovvero tutte quelle che restituiscono un file HTML.

4.4.1 application.properties

Path: T1-G11/applicazione/manvsclass/src/main/resources

Il file **application.properties** in un progetto Java Spring è utilizzato per configurare le proprietà dell'applicazione. Spring Boot, in particolare, utilizza questo file per definire vari parametri di configurazione dell'applicazione, come impostazioni del database, configurazioni di sicurezza, configurazioni del server e molte altre opzioni. Nel nostro caso, abbiamo aggiunto due voci per garantire che Thymeleaf, ovvero il motore di template utilizzato da Spring, cercasse i file nella directory giusta e con l'estensione .html .

```
spring.thymeleaf.prefix=classpath:/templates/  
spring.thymeleaf.suffix=.html
```

4.4.2 default.conf

Path: ui_gateway

Il file **default.conf** è una configurazione di Nginx, un server web e reverse proxy. Questa configurazione specifica come gestire le richieste in arrivo sulla porta 80 del server. Anche in questo caso, è stato necessario anteporre alle rotte il path /it e /en per permettere al web server di risolvere i nomi con cui ora vengono identificate le pagine.

```
server {  
    listen 80;  
    server_name localhost;  
    proxy_read_timeout 600s;  
    # Reindirizza la radice a /login  
    location = / {  
        return 301 /login;  
    }  
    # Gestisce le route che corrispondono a /loginAdmin, /  
    # registraAdmin, /orderbydate, /Dfilterby..., ...  
    location ~ ^/(loginAdmin|registraAdmin|orderbydate|Dfilterby.+|  
        orderbyname|delete|getLikes|uploadFile|t1) {  
        include /etc/nginx/includes/proxy.conf;  
        proxy_pass http://manvsclass-controller-1:8080;  
    }  
    # Gestisce le route che corrispondono a /it/home, /en/home, /it/  
    # home_adm, /en/home_adm, ...  
    location ~ ^/(it|en) /(home|home_adm|loginAdmin|registraAdmin|  
        modificaClasse|Reports|uploadClasse|reportClasse) {  
        include /etc/nginx/includes/proxy.conf;  
        proxy_pass http://manvsclass-controller-1:8080;  
    }  
    # Gestisce le route che corrispondono a /logout, /t23  
    location ~ ^/(logout|t23) {  
        include /etc/nginx/includes/proxy.conf;  
        proxy_pass http://t23-g1-app-1:8080;
```

```
}

# Gestisce le route che corrispondono a /it/login, /en/login, /it/
# register, /en/register, ...
location ~ ^/(it|en)/(login|register|mail_register|password_change|
|password_reset) {
    include /etc/nginx/includes/proxy.conf;
    proxy_pass http://t23-g1-app-1:8080;
}

#Gestisce i percorsi relativi alle richieste POST
location ~ ^/(login|register|mail_register|password_change|
|password_reset) {
    include /etc/nginx/includes/proxy.conf;
    proxy_pass http://t23-g1-app-1:8080;
}

# Gestisce la route che corrisponde a /t5
location ~ ^/(t5) {
    include /etc/nginx/includes/proxy.conf;
    proxy_pass http://t5-app-1:8080;
}

# Gestisce le route che corrispondono a /it/main, /en/main, /it/
# editor, /en/editor, /it/report, /en/report, ...
location ~ ^/(it|en)/(main|editor|report) {
    include /etc/nginx/includes/proxy.conf;
    proxy_pass http://t5-app-1:8080;
}

# Gestisce le richieste che iniziano con /api
location ^~ /api {
    include /etc/nginx/includes/proxy.conf;
    proxy_pass http://api_gateway-gateway-1:8090;
}
```

```
# Disabilita i log di accesso
access_log off;
# Specifica il percorso per il log degli errori
error_log /var/log/nginx/error.log error;
}
```

Osservazione : Per l'aggiunta di ulteriori lingue, è sufficiente aggiungere il carattere pipe | seguito dalle due lettere identificative della nuova lingua (ad esempio, |es|fr per aggiungere lo spagnolo e il francese).

4.5 Soluzione alternativa 1

Si è approcciato il problema anche attraverso un metodo meno invasivo dal punto di vista del backend, che provasse a gestire la traduzione senza alterare la struttura delle directory "templates", e preservando di conseguenza le rotte esistenti e le configurazioni del reverse proxy. Tale soluzione prevede l'utilizzo di alcuni servizi esterni, attraverso API pubbliche, per effettuare la traduzione del file HTML sulla base della posizione geografica dell'utente, rilevata attraverso le coordinate geografiche e/o l'indirizzo IP del client. Nello specifico, i servizi in questione sono:

- Geolocation DB, che restituisce un insieme di informazioni relative alla posizione, tra cui un codice identificativo della nazione (country_code, ad esempio "IT");
- DeepL API, fornite dall'applicazione di traduzione online basata su Machine Learning, le quali permettono di impostare un attributo nella POST request (tag_handling) che nasconde al traduttore i tag HTML e XML.

Tuttavia, questa soluzione ha presentato da subito alcune criticità. In primo luogo, le DeepL API hanno un limite di utilizzo mensile, anche nella loro versione a pagamento,

rendendo questa soluzione costosa e poco scalabile. In secondo luogo, la soluzione è pensata per eseguire lo script di traduzione una sola volta, al caricamento della pagina, e modificare il contenuto del file una volta sopraggiunta la risposta. Ciò non si è rivelato effettivamente realizzabile, in quanto i file HTML hanno una struttura tra loro disomogenea, e uno script che sia abbastanza generico da essere riutilizzabile per tutti i file richiede una modifica del contenuto che coinvolge l'intero tag **<body>**, operazione che causa il reload della pagina e quindi la chiamata in cascata dello stesso script, in un ciclo infinito. Per ovviare a questo problema, esistono varie soluzioni:

- dotare il body di ogni file html di un **<div class="external-container">**, ovvero di un wrapper per i contenuti della pagina, in modo da effettuare su di esso la modifica senza sollecitare una ricarica;
- inserire un **<button class="translate">**, al cui click si aziona la traduzione in maniera non automatica, e sul quale è possibile applicare la funzione *event.preventDefault()*, che impedisce alla pagina di ricaricarsi;
- utilizzare un framework come React, pensato per realizzare SPA (Single Page Application), ovvero applicazioni web basate su componenti che aggiornano i contenuti in maniera dinamica, senza causare alcun reload della pagina principale, che resta appunto unica.

L'adozione di una tale tecnologia è fortemente consigliata in caso di refactoring futuri dell'applicazione in quanto, seppur a seguito di un lavoro oneroso, permetterebbe di creare pagine tra loro omogenee e stilisticamente accattivanti grazie al riuso dei componenti, migliorando la UX (User eXperience), e sollevando il backend dalla maggior parte dei compiti di routing, il quale verrebbe gestito direttamente lato client da alcuni componenti della libreria "router" di React. Tutto ciò avrebbe un forte impatto benefico anche sulla complessità e sulla leggibilità del codice backend, in quanto andrebbero preservate unicamente le rotte che riguardano le API, ovvero le operazioni di

calcolo e di interazione con il database, mentre potrebbero essere rimosse le rotte che restituiscono pagine HTML.

Di seguito è presentato un esempio di come si potrebbe realizzare la seconda soluzione, con il bottone di traduzione, con l'utilizzo della libreria jQuery.

```
var getCountryCode = () => {
    return new Promise((resolve, reject) => {
        const apiKey = 'hidden-for-privacy';
        const url = 'https://geolocation-db.com/json/${apiKey}';

        $.getJSON(url)
            .done((data) => {
                resolve(data.country_code);
            })
            .fail((error) => {
                reject(error);
            });
    });
};

var main = async () => {
    "use strict";

    const url = 'http://localhost:8080/translate';

    try {
        const countryCode = await getCountryCode();
        console.log(countryCode);
    }
}
```

```
const textToTranslate = [$(‘head’).html(), $(‘body’).html()];  
  
const payload = {  
    text: textToTranslate,  
    tag_handling: ‘html’,  
    target_lang: countryCode  
}  
  
$.ajax({  
  
    url: url,  
    method: ‘POST’,  
    dataType: ‘json’,  
    contentType: ‘application/json’,  
    data: JSON.stringify(payload),  
  
    success: (response) => {  
        $(‘head’).html(response.translations[0].text);  
        $(‘body’).html(response.translations[1].text);  
    },  
  
    error: (error) => {  
        console.error(‘Errore nella richiesta POST:’, error);  
    }  
});  
  
}  
} catch (error) {  
    console.error(‘Errore durante il recupero del codice del paese
```

```
:', error);  
}  
};  
  
$(document).ready(main);
```

La rotta `"/translate"` effettua una chiamata al server che rigira il payload al servizio DeepL API, bypassando le problematiche dovute al CORS (Cross Origin Resource Sharing), limitazione adottata dalla suddetta API che impedisce l'accesso al servizio da parte di richieste provenienti dal browser.

4.6 Soluzione alternativa 2

La soluzione più semplice, sebbene parzialmente limitata dal browser utilizzato e dalle impostazioni ad esso associate, è quella di sfruttare la potenza espressiva di HTML ed i servizi di traduzione automatica forniti da alcuni browser (es. Google Chrome). Infatti, dotando l'attributo `<html>` di un attributo `<html lang="it">`, si fornisce al browser un metadato relativo alla lingua del testo inserito all'interno dei tag successivi. Chrome effettua automaticamente un confronto tra tale lingua e quella di default, scelta nelle impostazioni, ed eventualmente propone una scelta tra la versione in lingua originale e quella generata automaticamente a seguito della traduzione.

Chapter 5

Organizzazione del Lavoro con Scrum

5.1 Introduzione a Scrum

Scrum è un framework Agile focalizzato sulla gestione dei progetti e lo sviluppo del software. Basato su principi e valori definiti nel "Manifesto Agile," Scrum fornisce una struttura per aiutare i team a lavorare in modo collaborativo.

5.1.1 Adozione di Scrum nel Progetto

Nel contesto del progetto, abbiamo adottato il framework Scrum come metodologia di gestione e sviluppo per l'integrazione di nuove funzionalità. Il progetto si è sviluppato in tre iterazioni, con un numero variabili di sprint, per una durata complessiva di circa 2 mesi di lavoro. Ogni iterazione è iniziata con un Iteration Planning, durante il quale abbiamo creato le User Stories e identificato quali implementare nell'iterazione corrente. Ogni Sprint ha avuto un proprio Sprint Planning, durante il quale abbiamo suddiviso le User Stories in task assegnati ai membri del team. Abbiamo utilizzato piattaforme

come Notion e Miro per tracciare i task.

5.2 User Stories Totali

Le User Stories descrivono parti di funzionalità desiderate dal punto di vista dell'attore coinvolto, guidando lo sviluppo nel processo agile. Le User Stories ottimali seguono il paradigma INVEST:

- **Indipendenti:** Le User Stories devono essere indipendenti per evitare stime errate e vincoli di implementazione.
- **Negoziabili:** Le storie catturano l'essenza di una richiesta; i dettagli sono negoziati tra cliente, product owner e team.
- **Di Valore:** È essenziale interrogarsi sul vero beneficiario della User Story per evitare di perdere tempo in storie di valore per nessuno.
- **Stimabili:** La stima è fondamentale; in caso di incertezza, si può avviare un'analisi per stimare le attività.
- **Della Giusta Dimensione:** Le storie devono avere la dimensione adeguata; è opportuno granulare le storie troppo grandi o combinare più storie piccole.
- **Testabili:** Le User Stories devono includere criteri di accettazione misurabili attraverso test.

5.2.1 Paradigma delle 3C

Le User Stories seguono il paradigma delle 3C: Card, Conversation, Confirmation.

- **Card (Carta):** Rappresenta la User Story con una breve descrizione del comportamento desiderato del sistema.

- **Conversation (Conversazione):** Si riferisce alle discussioni dettagliate tra stakeholder e membri del team per chiarire e approfondire la comprensione della User Story.
- **Confirmation (Conferma):** Definisce i criteri di accettazione che indicano quando la User Story è completata correttamente.

Le User Stories finali sono descritte nei capitoli 3 e 4.

5.2.2 Criteri di Accettazione

I criteri di accettazione sono scritti utilizzando il modello "Given-When-Then," garantendo allineamento e chiarezza nella definizione delle funzionalità. Ogni criterio fornisce condizioni che devono essere soddisfatte per considerare una User Story completata con successo.

Di seguito sono riportati i criteri di accettazione per ogni storia creata.

5.3 Prima iterazione

Durante la prima iterazione sono stati svolti i task relativi alla analisi di documentazione e revisione del codice relativi a vari componenti del progetto, in particolare il componente T1 dal sottogruppo A1 e il componente T8 dal sottogruppo A8.

5.4 Seconda iterazione

Dopo una riunione di gruppo in cui abbiamo effettuato una prima installazione e visto il funzionamento della web app, abbiamo scritto la seguente To-Do list:

Seconda iterazione

To-do list

- Far partire l'applicazione e verificarne il workflow.
- Capire quali rotte espone il server per effettuare il caricamento (com'è allo stato attuale).
- Capire qual è la serie di chiamate effettuate, a partire dalla ricezione della richiesta fino alla generazione dei test e il salvataggio.
- Capire se il requisito richiede che il caricamento della classe e dei relativi test sia contestuale, o se sia previsto un caricamento separato.
- Capire come, all'interno del filesystem, vadano memorizzate le classi ed i test.
- Realizzare una rotta alternativa che repliche il funzionamento di quella esistente, eccetto la chiamata ai Robot per la generazione dei test, che saranno contenuti nel richiesta HTTP.
- Realizzare una barra di caricamento per fornire riscontro dei progressi.

In particolare i Task relativi al requisito R6 sono :

To-do list - R6

- Aggiunta di una variabile di configurazione che indica la lingua.
- Modificare le rotte in modo da avere
 - /it/rotta per la versione in italiano
 - /en/rotta per la versione in inglese

Sebbene ritenuti meno prioritari del requisito R4 è stato deciso di svolgerli entrambi durante la seconda iterazione e operando in parallelo una revisione e refactoring per il

task R4 in quanto fosse necessaria effettuarla con un vincolo di precedenza rispetto agli altri task di R4 che sono poi oggetto della terza iterazione.

Tasks

 Incontro per documentare le modifiche apportate al progetto alla fine di ogni iterazione.

Seconda iterazione

- [1 persona] - Tradurre i file HTML in inglese.
- [2 persone] - Individuare le rotte che restituiscono le pagine HTML, e sostituire quelle in italiano con la rotta `/it/{oldRoute}`, aggiungendo la rotta `/en/{oldRoute}` per quelle in inglese.
- [3 persone] - Refactoring (nomi variabili e commenti) alla funzione critica di salvataggio e a quelle da essa chiamate.

Terza iterazione

- [3 persone] - Creare una nuova rotta e una nuova UI per la funzionalità R4.
- [3 persone] - Implementazione della funzionalità R4 vera e propria, a partire dalla funzione già esistente di cui si è documentato il funzionamento.

Il backlog relativo alla seconda iterazione era il seguente:

CHAPTER 5. ORGANIZZAZIONE DEL LAVORO CON SCRUM



Figure 5.1: Backlog seconda iterazione

5.5 Terza iterazione

Come evidenziato anche dall'ultima immagine, al terza iterazione si è concentrata tutta sulla implementazione e valutazione del funzionamento del requisito R4 con il completamento dei relativi tasks. Prima di iniziare la situazione era la seguente:

CHAPTER 5. ORGANIZZAZIONE DEL LAVORO CON SCRUM

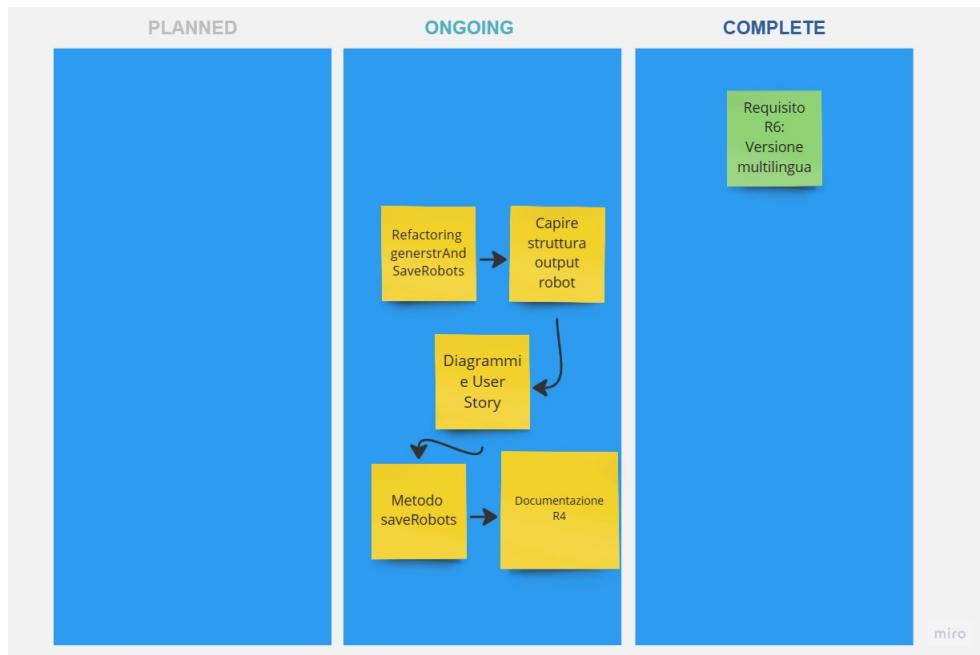


Figure 5.2: Backlog terza iterazione

In seguito al nostro lavoro tutti i task sono stati completati e il backlog risultà così completo:

CHAPTER 5. ORGANIZZAZIONE DEL LAVORO CON SCRUM

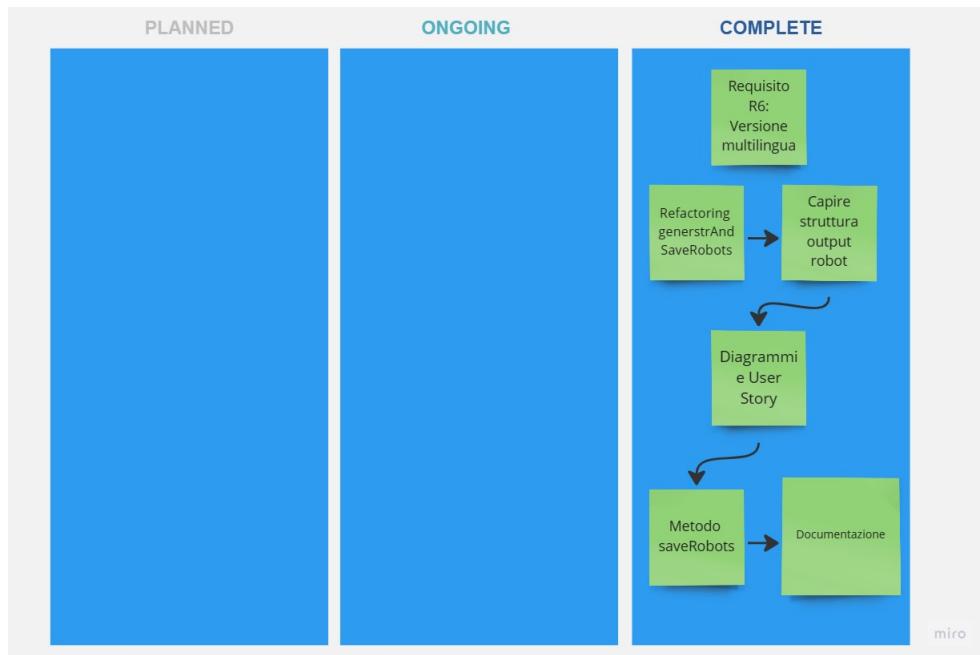


Figure 5.3: Backlog finale

Chapter 6

Guida all'installazione

6.1 Docker e WSL

Docker è una piattaforma di containerizzazione che consente la creazione, la distribuzione e la gestione efficiente e isolata di applicazioni. Tutto ciò assicura la massima portabilità dell'applicazione software, indipendentemente dall'ambiente di esecuzione sottostante. Per effettuare l'esecuzione di Docker in ambiente Windows è necessario l'impiego della cosiddetta WSL, ossia la Windows Subsystem for Linux. Essa permette di eseguire un ambiente Linux all'interno di Windows, consentendo agli utenti di utilizzare strumenti e applicazioni Linux direttamente sul proprio sistema operativo Windows. L'utilizzo di Docker permette di creare immagini e container che operano in ambienti isolati, consentendo l'esecuzione di servizi separati e la comunicazione tra di essi attraverso il mapping dei porti specifici. L'approccio di Docker quindi, favorisce la modularità, la scalabilità e semplifica la distribuzione delle applicazioni, ottimizzando l'efficienza e la gestione delle risorse. È stato possibile includere un file system locale all'interno del container, utilizzando percorsi assoluti per accedervi in combinazione con l'artefatto prodotto. Da questa configurazione è stata creata un'istanza di macchina virtuale, favorendo l'utilizzo

delle risorse del file system senza dovervi rinunciare.

6.2 Installazione

6.2.1 Passo 1

Scaricare Docker Desktop (9.4) per il proprio sistema operativo. N.B. Si potrebbe riscontrare all'avvio di Docker, l'errore 136 Docker desktop - unexpected wsl error. In tal caso eseguire nel terminale il comando wsl - -shutdown e riavviare la macchina. Se neanche questo dovesse funzionare, si dovrebbe installare WSL (wsl - -install sul terminale) o aggiornarlo all'ultima versione.

6.2.2 Passo 2

Per Windows - avviare lo script file installer.bat. Per MacOS - eseguire nella cartella dove è presente il file installermac.sh il comando chmod +x installermac.sh per renderlo eseguibile, e poi ./installermac.sh per eseguirlo. Tale installazione porterà alla:

1. Creazione della rete global-network comune a tutti i container
2. Creazione del volume VolumeT9 comune ai Task 1 e 9 e del VolumeT8 comune ai Task 1 e 8.
3. Creazione dei singoli container nel Docker desktop.

Alla fine dell'installazione, bisognerà avviare tutti i container che non si presentano attivi, tranne ui gateway, il quale dovrà essere avviato solo dopo l'attivazione di tutti gli altri. N.B: il container relativo al Task 9 (Progetto-SAD-G19-master) si sosponderà autonomamente dopo l'avvio. Esso viene utilizzato solo per popolare il volume VolumeT9 condiviso con il Task 1.

6.2.3 Passo 3

Si deve configurare il container manvsclass-mongo db-1 così come descritto anche nella documentazione del Task 1. Per fare ciò bisogna fare le seguenti operazioni:

1. Posizionarsi in Docker Desktop all'interno del terminale del container
2. Digitare il comando "mongosh"
3. Digitare successivamente ed in maniera sequenziale i seguenti comandi;

```
use manvsclass
db.createCollection(ClassUT);
db.createCollection(interaction);
db.createCollection(Admin);
db.createCollection(Operation);v
db.ClassUT.createIndex( difficulty: 1 )
db.Interaction.createIndex( name: text, type: 1 )
db.interaction.createIndex( name: text )
db.Admin.createIndex(username: 1)
```

Una volta effettuati tutti i passaggi precedentemente definiti l'intera applicazione è pienamente configurata e raggiungibile sulla porta :80

6.2.4 Utilizzo

Dopo aver completato l'installazione e avviato i container (importante accertarsi che il container **t23-g1** sia avviato per poter far partire l'UI senza che questa termini dopo pochi secondi), è ora possibile procedere con la registrazione tramite la schermata di accesso. Tuttavia, prima di iniziare a giocare, è necessario inserire una classe di test da sottoporre a verifica. Questa operazione può essere eseguita esclusivamente dall'utente amministratore. Pertanto, è essenziale che l'amministratore si registri al sistema in modo da poter successivamente caricare la classe di test. Per semplificare il processo, di seguito

CHAPTER 6. GUIDA ALL'INSTALLAZIONE

sono fornite le istruzioni dettagliate su come effettuare la registrazione e caricare la classe di test attraverso una procedura guidata

Registrazione Admin

L' amministratore, se è il primo accesso alla applicazione, si deve registrare nella schermata di registrazione ottenuta inserendo nella barra degli indirizzi questo url:

`http://localhost/registraAdmin.`

Apparirà pagina web in cui è possibile inserire il proprio nome, cognome, username e password.

Registrazione utente

Se l'utente è al primo utilizzo dell' applicazione, deve digitare l'url `http://localhost/login` per accedere alla schermata di login, e cliccare sulla dicitura Non sei ancora registrato? Registrati. A questo punto è possibile effettuare la registrazione inserendo, il proprio nome, cognome, email, e password.

N.B: La password deve contenere almeno un carattere speciale, una lettera maiuscola, una minuscola, e deve contenere un minimo di 8 caratteri. Una volta registrato e loggato correttamente, l'utente può scegliere, tra le classi di test caricate dall'amministratore, quale sottoporre al sistema e finalmente, iniziare a giocare per provare a battere il robot, e raggiungere un livello di copertura del test maggiore!

Chapter 7

Glossario

7.1 Robot

Nel contesto di questa applicazione, il termine "Robot" si riferisce a strumenti di generazione automatica di test, come Randoop o Evosuite. Questi rappresentano una sfida per gli studenti di Software Testing che utilizzano l'applicazione.

7.2 Turno

Ogni volta che un giocatore avvia una nuova partita (Game), deve selezionare il "robot" e la classe da sottoporre a test (round). Ogni tentativo di sottoporre un risultato viene chiamato "turno" o "tentativo".

7.3 Giocatore

L'utente registrato, denominato "giocatore" o "Player" in questo documento, partecipa attivamente all'applicazione.

7.4 Admin

Un utente registrato come amministratore, denominato "Admin", partecipa attivamente all'applicazione potendo introdurre Classi e Test nel sistema.

7.5 Docker

Docker è una piattaforma software che facilita la creazione, il test e la distribuzione rapida di applicazioni. Utilizza container standardizzati contenenti tutto il necessario per l'esecuzione corretta, inclusi librerie, strumenti di sistema, codice e runtime. Docker consente di distribuire e adattare risorse in qualsiasi ambiente, mantenendo il controllo sul codice eseguito. Nell'applicazione, Docker è essenziale per l'integrazione e la comunicazione tra task implementati con diverse tecnologie.

7.6 Notion

Notion è una piattaforma di gestione delle attività . Notion è noto per la sua interfaccia user-friendly e flessibilità, utilizzato per gestire progetti, pianificare attività personali e anche tracciare lo sviluppo software.

7.7 Miro

Miro è una piattaforma di collaborazione online con lavagna virtuale e strumenti di collaborazione. Favorisce la creazione e la condivisione di diagrammi, mappe concettuali, wireframe e altri contenuti visivi per agevolare la comunicazione e la collaborazione tra membri del team, anche in posizioni geografiche diverse.