

UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II



SCUOLA POLITECNICA E DELLE SCIENZE DI BASE

DIPARTIMENTO DI INGEGNERIA ELETTRICA E TECNOLOGIE
DELL'INFORMAZIONE

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

Anno Accademico 2024-25

Progetto d'esame – Software Architecture Design

DOCUMENTAZIONE SAD SECURITY



STUDENTI

Antonio Napolitano

M63001464

Sabatino Veturo

M63001477

DOCENTE

Prof. Anna Rita Fasolino



INDICE

1. INTRODUZIONE	4
2. SPECIFICHE DEL PROGETTO	5
2.3 Requisiti Funzionali	6
2.4 Requisiti non funzionali	6
2.5 Visione ad alto livello del sistema	7
3. STRUMENTI UTILIZZATI	12
3.1 Strumenti di sviluppo	12
3.2 Tecnologie utilizzate	12
4. FASE DI ANALISI	13
4.1 Use case diagram	14
4.2 Sequence diagram e scenari d'uso	15
4.2.1 Registrazione Studente	16
4.2.2 Login Studente	17
4.2.3 Login Professore	18
4.2.4 Cambio Password	19
4.2.5 Iscrizione Classe	20
4.2.6 Avvio Room	21
4.2.7 Creazione Room	22
4.2.8 Creazione Classe	23
4.2.9 Visualizza Risultati	23
4.2.10 Inserimento Flag	24
4.2.11 Visualizzazione Studenti	25
4.2.12 Visualizzazione Room	25
4.3 Activity Diagram	26
4.3.1 Avvio Laboratorio	27
4.3.2 Cambio Password	28
4.3.3 Crea Laboratorio	29
4.3.4 Inserimento Flag	30
4.4 Class diagram	31
4.5 Entity Relation Diagram	32
5. FASE DI PROGETTAZIONE	33
5.1 Component diagram	34
5.2 Package diagram	36
5.3 Deployment diagram	37
6. IMPLEMENTAZIONE	39



6.1 Sequence diagram di dettaglio	39
6.1.1 Crea laboratorio	39
6.1.2 Avvia laboratorio	41
6.2 Interfacce	43
7. TESTING	45
7.1 Test suite per le Rest API	45
7.2 Testing con Postman	48
7.2.1 Testing creaRoom	48
7.2.2 Testing creaRoom	55
8. CONTAINERIZZAZIONE	58
9. SVILUPPI FUTURI	59
A. Guida agli sviluppatori	60
B. Guida alla scrittura dei laboratori per i Professori	61



1. INTRODUZIONE

L'applicazione che verrà presentata nella seguente documentazione è denominata SAD Security e nasce dall'idea di avvicinare lo studio delle discipline IT quanto più possibile alla dimensione pratica e reale partendo dal punto di vista che discipline di questo tipo, oltre a prestarsi alla ricerca, nascono per analizzare e risolvere problemi reali.

In questo contesto specifico contesto il focus dell'applicazione è il campo della sicurezza informatica ma si può pensare di generalizzare il discorso a tutte quelle che sono le tecnologie dell'ambito dell'ingegneria che prevedono una specifica applicazione sul campo.

La seconda osservazione che ha dato vita a questo progetto riguarda la possibilità di modificare il punto di vista con cui si approccia lo studio andandone ad evidenziare i punti di contatto che possono esserci con attività finalizzate al gaming; tutto ciò che può essere trasformato in un gioco con un obiettivo, magari un punteggio, e degli avversari da battere o delle missioni da completare permette di creare un clima più favorevole, e quindi più produttivo, a prescindere dal campo specifico di applicazione.

SAD Security è una webapp che permette di unire questi due concetti dando la possibilità di imparare giocando proponendosi come strumento di e-learning di supporto agli studi in un contesto didattico in cui i docenti propongono agli studenti delle challenge che possono essere risolte in piena autonomia; gli unici strumenti richiesti sono un qualsiasi strumento informatico che abbia connessione ad internet e la possibilità di eseguire una connessione SSH.

La gestione della piattaforma viene fatta completamente in browser sia per quanto riguarda gli admin (i docenti che si occuperanno di fornire le challenge) che per gli studenti che intendono utilizzare questo strumento.

Nel resto del documento verranno forniti dettagli che riguardano sia la struttura complessiva dell'applicazione (attraverso diagrammi e tabelle) che procedure rivolte agli amministratori oltre che ai programmatori interessati ad espandere e migliorare questo progetto che ben si presta alla collaborazione ed alla condivisione della conoscenza.



2. SPECIFICHE DEL PROGETTO

In questa sezione verranno forniti i dettagli relativi ai requisiti assegnati, agli utenti del sistema ed a tutti quei dettagli che vengono definiti nelle prime fasi dello sviluppo di un progetto.

Per i dettagli progettuali e tecnici si rimanda ai capitoli successivi del documento.

2.1 Utenti del sistema

Gli utenti che interagiscono col sistema sono di due tipi

- **Professori:** Sono gli attori che rendono possibili la maggior parte delle operazioni degli studenti. Hanno il compito di creare le classi, alle quali gli studenti possono iscriversi, e di rendere disponibili le challenge agli studenti di ciascuna classe.
- **Studenti:** Sono i principali protagonisti della piattaforma. Possono iscriversi alle varie classi, avviare e completare i laboratori oltre che consultare i risultati degli altri utenti.

2.2 Glossario dei termini

In questa sezione viene specificato il significato dei termini che verranno utilizzati nel resto della documentazione in modo tale da non creare fraintendimenti o errori di interpretazione legati all'uso equivalente di parole che si riferiscono agli stessi concetti.

Essendo la documentazione indirizzata a diverse figure (utilizzatori, programmatori, curiosi), verranno riportati termini, sinonimi e significati.

Concetto	Sinonimo	Descrizione
Laboratorio	Challenge, Room, Container	Rappresenta la sfida che gli studenti sono chiamati a risolvere.
Admin	Professore	Rappresenta l'attore Professore che ha il compito di gestire la piattaforma a livello di interfaccia grafica.
Utente	Studente	Usato per indicare un attore Studente anche se non ancora registrato.
Avviare, Stoppare Room	Distruggere container	Azione con la quale viene creata/interrotta un'istanza univoca della challenge. In sostanza permette allo studente di cominciare la sfida o interromperla.
Creare Room	-	Azione del professore che permette di caricare una nuova challenge sulla piattaforma.
Flag	-	Prova del completamento di una challenge che consiste in una stringa testuale.
Piattaforma	Webapp, Applicazione, Sistema	Rappresentano dei termini con cui ci si riferisce al sistema inteso nella sua totalità. Questi termini vengono usati soprattutto nei casi in cui non è importante descrivere quale parte dell'applicazione sta eseguendo ma cosa sta facendo.
Operazioni di management	-	Rappresentano le operazioni CRUD. Non è stato utilizzato il termine CRUD in quanto non sempre ne è stata rispettata la convenzione.



2.3 Requisiti Funzionali

I requisiti funzionali sono le specifiche delle funzionalità e dei comportamenti attesi che un sistema software deve essere in grado di fornire. In altre parole, i requisiti funzionali descrivono ciò che il software deve fare per soddisfare le esigenze dell'utente o del cliente.

I seguenti requisiti sono frutto della sintesi fra le idee del team e quelle della docente che ha seguito lo sviluppo fin dalle prime sedute di brainstorming.

- **REQF#01** - Lo studente deve poter effettuare il login sulla piattaforma.
- **REQF#02** - Lo studente deve poter effettuare la registrazione sulla piattaforma.
- **REQF#03** - Lo studente deve essere in grado di effettuare l'iscrizione ad una classe.
- **REQF#04** - Lo studente deve poter scorrere la lista delle room disponibili per una determinata classe.
- **REQF#05** - Lo studente deve avere la possibilità di avviare un'istanza di un laboratorio disponibile.
- **REQF#06** - L'applicazione deve fornire un comando allo studente per una connessione esterna con la room.
- **REQF#07** - Il professore può creare classi.
- **REQF#08** - Il professore può creare nuove room da associare ad una classe.
- **REQF#09** - Lo studente deve avere la possibilità di stoppare una room in esecuzione.
- **REQF#10** - L'applicazione deve permettere allo studente l'inserimento della chiave di completamento delle room.
- **REQF#11** - Il sistema deve pubblicare in automatico i risultati relativi ad una room

2.4 Requisiti non funzionali

I requisiti non funzionali sono le specifiche che riguardano aspetti diversi dalle funzionalità del software. Mentre i requisiti funzionali si riferiscono al "cosa" il software deve fare, i requisiti non funzionali si riferiscono al "come" il software deve essere.

- **REQNF#01** - Il limite di laboratori eseguibili in contemporanea per uno studente è pari ad uno.
- **REQNF#02** - L'applicazione deve essere intuitiva.
- **REQNF#03** - L'applicazione deve fornire informazioni circa gli errori.



2.5 Visione ad alto livello del sistema

Per cominciare a dare un'idea di come la webapp funzioni, viene fornito il diagramma UML denominato *context diagram* utile a mostrare, in modo molto semplice e ad alto livello, come un sistema interagisce con il suo ambiente esterno.

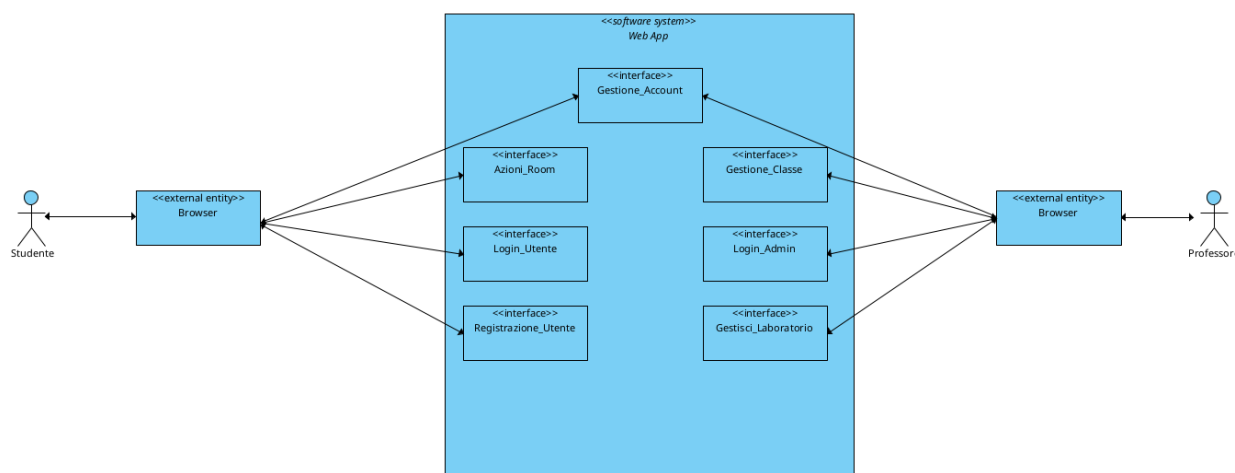


Figura 2.1: Context Diagram

Prendendo spunto dalla figura si osserva:

- **Studente**
 - *Gestione_Account*: Interfaccia che permette all'utente di effettuare le operazioni di account management.
 - *Azioni_Room*: Interfaccia che permette all'utente di effettuare le operazioni di interazione con le challenge.
 - *Login_Utente*: Interfaccia che permette all'utente di effettuare il login.
 - *Registrazione_Utente*: Interfaccia che permette all'utente di effettuare una registrazione.
- **Professore**
 - *Gestione_Account*: Interfaccia che permette all'admin di effettuare le operazioni di account management.
 - *Login_Admin*: Interfaccia che permette all'admin di effettuare il login.
 - *Gestione_Laboratorio*: Interfaccia che permette all'admin di effettuare le operazioni di management delle challenge.
 - *Gestione_Classe*: Interfaccia che permette all'admin di effettuare le operazioni di management delle classi.

In conclusione si riportano alcuni screenshot delle interfacce descritte in modo da presentare in maniera concreta quanto descritto finora.

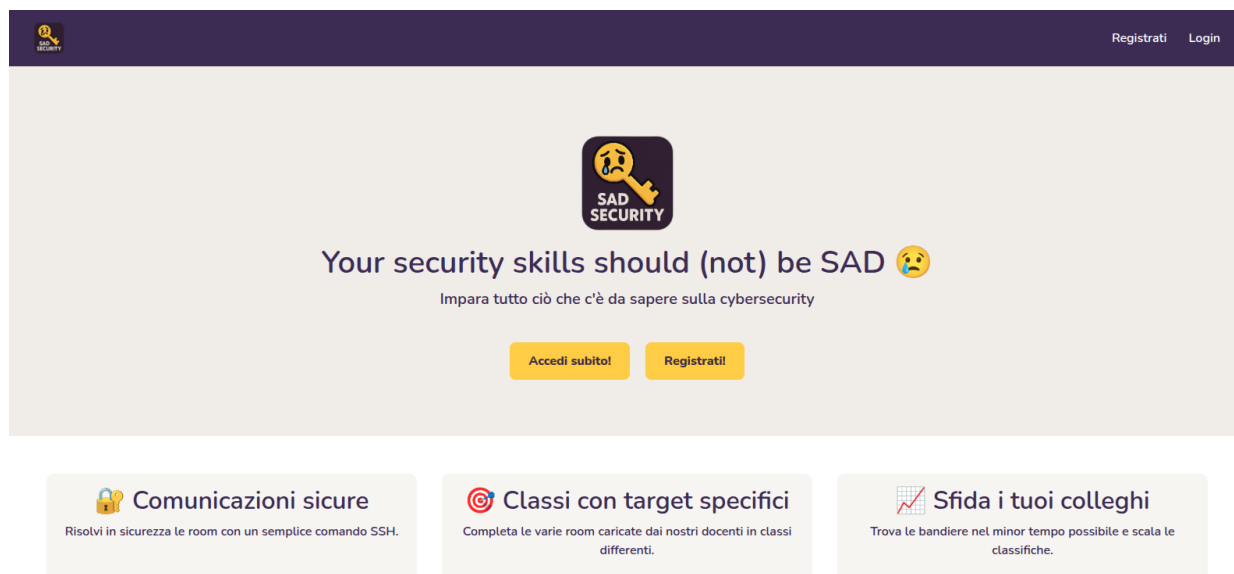


Figura 2.2: Landing Page

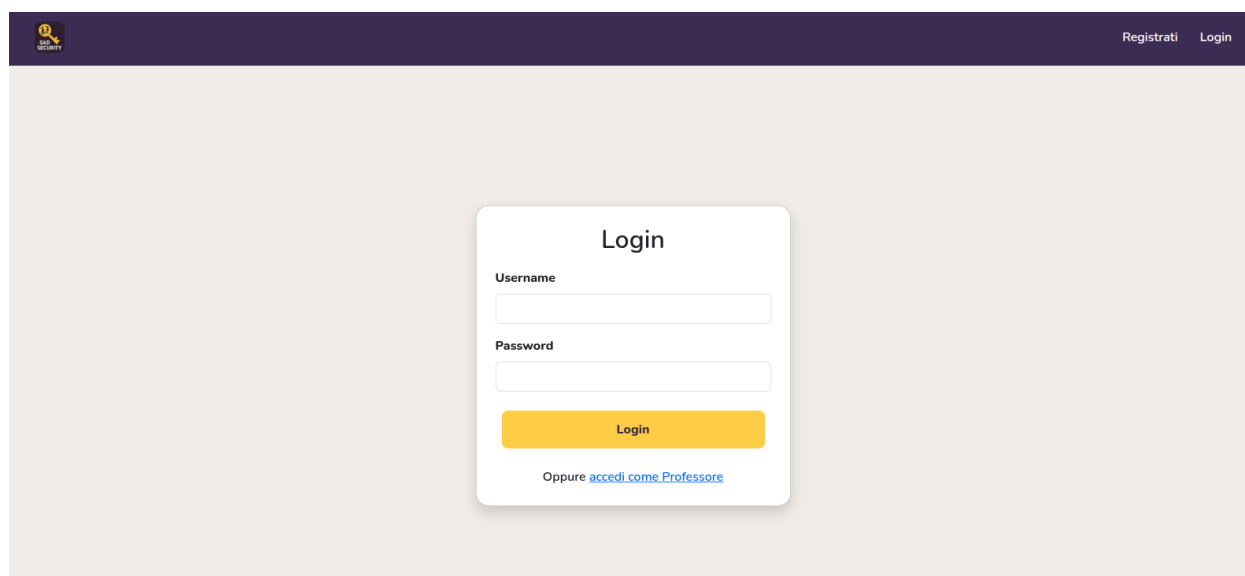



Figura 2.3: Login Studente



 Registrati Login

Registrati


Username

E-Mail

Password

Registrati

Figura 2.4: Registrazione Studente

 Benvenuto, bob!

Classi a cui sei iscritto

SAD

Operazioni

Risultati Room

Impostazioni

Iscriviti a una Classe

Cambia Password

Logout

Benvenuto Studente

Seleziona un'opzione dalla barra laterale per iniziare.

© 2025 Sad Security. Tutti i diritti riservati.

Figura 2.5: Dashboard Studente

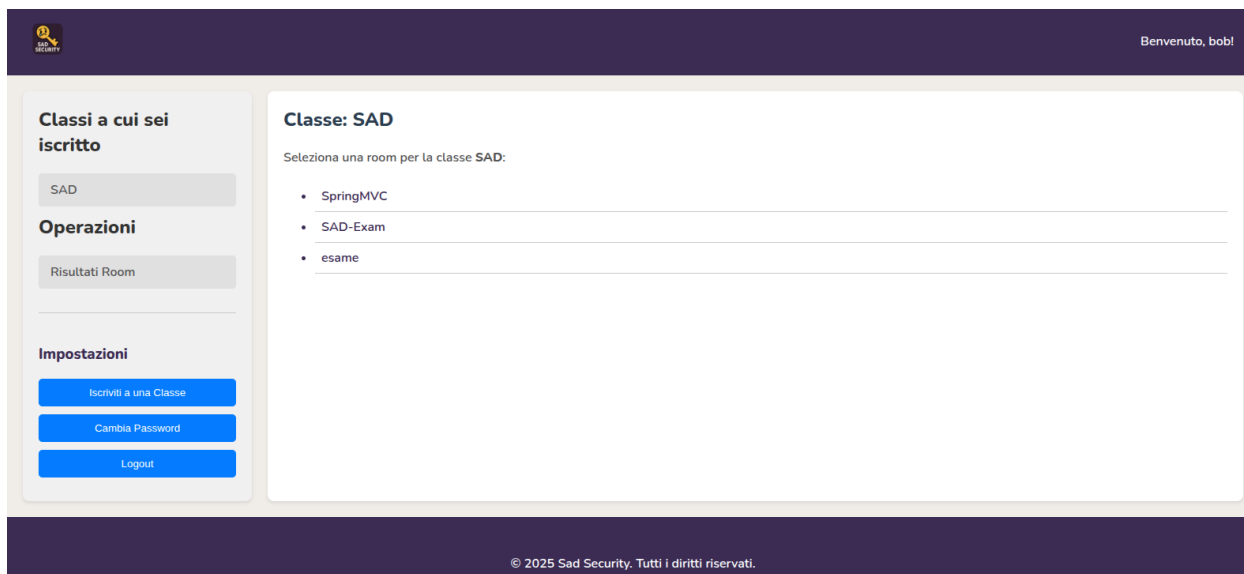


Figura 2.6: Azioni_Room

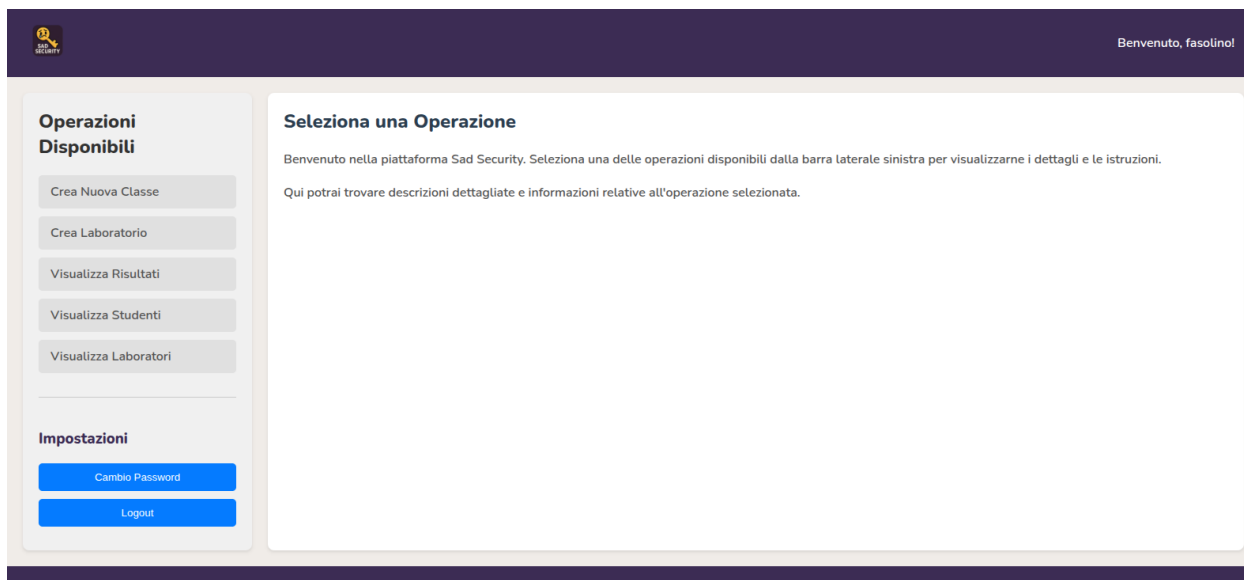



Figura 2.7: Dashboard Professore



Benvenuto, fasolino!

Operazioni Disponibili

Crea Nuova Classe

Crea Laboratorio

Visualizza Risultati

Visualizza Studenti

Visualizza Laboratori

Impostazioni

Cambio Password

Logout

Crea Laboratorio

Crea un nuovo laboratorio per una classe inserendo il nome, una breve descrizione (255 caratteri), la flag di completamento ed il docker compose file.

Nome classe:

Seleziona una classe

Nome Laboratorio:

Descrizione Laboratorio:

Flag:

File YAML:

Scegli file | Nessun file selezionato

Crea Laboratorio

Figura 2.8: Schermata di creazione del laboratorio



3. STRUMENTI UTILIZZATI

In questa sezione verranno presentate le tecnologie e gli strumenti utilizzati per lo sviluppo dell'applicazione.

3.1 Strumenti di sviluppo

Per quanto riguarda i meccanismi di collaborazione sono stati utilizzati:

- Strumenti di comunicazione real-time Discord e Microsoft Teams per la condivisione dello schermo durante le sedute di brainstorming e di allineamento durante lo sviluppo da remoto.
- Come strumenti di condivisione dei file è stato utilizzato Obsidian con un vault condiviso tramite Dropbox in modo tale da poter lavorare, anche in diretta, sugli stessi documenti senza correre il rischio di trovarsi de-sincronizzati.
- Si è scelto di utilizzare GitHub come repository remota sfruttando le innumerevoli integrazioni nei framework che hanno semplificato alcune operazioni di routine.

Per quanto riguarda la produzione degli artefatti:

- Si è scelto di utilizzare Visual Paaradigm utilizzano la funzione Team grazie alla quale le operazioni di condivisione sono state semplificate dalla possibilità di avere una repository condivisa e strumenti UI.

Per quanto riguarda la scrittura del codice ed il deployment:

- Si è scelto di usare Docker per il deployment dell'applicazione che è stata scritta e pensata per essere distribuita e successivamente containerizzata per simulare una vera e propria situazione reale di routing anche se in ambiente di testing.
- Per la scrittura del codice si è scelto di utilizzare VsCode in funzione dalla sua versatilità e delle tantissime integrazioni che hanno permesso di coordinare diverse tecnologie con semplicità.

3.2 Tecnologie utilizzate

Per quanto riguarda la scelta delle tecnologie utilizzate si è optato per:

- Spring MVC per il core del server di backend insieme al framework Spring Security per quanto riguarda la parte relativa alla sicurezza.
- Python per una parte del backend che si occupa di operare direttamente con il CLI attraverso. L'uso di due tecnologie differenti è stata possibile grazie alla separazione logica delle varie componenti avvenuta nelle fasi precedenti a quella di implementazioni che ha permesso di scegliere la tecnologia più adatta allo scopo
- Per il DB si è scelto di operare con Postgres che ha offerto robustezza e semplicità d'uso grazie all'interfaccia grafica PgAdmin.
- Il front-end è stato completamente realizzato con HTML e JavaScript utilizzando per lo style codice CSS e Bootstrap.



4. FASE DI ANALISI

In questo capitolo verranno mostrati i cosiddetti diagrammi di analisi prodotti con Visual Paradigm che rappresentano una tipologia di diagrammi UML grazie ai quali è possibile documentare le prime fasi dello sviluppo del software.

Obiettivo di questa fase è quello di ragionare sui requisiti e di comprendere cosa è richiesto al sistema in termini di operazioni, interazioni e caratteristiche; in questa fase vengono anche formalizzati gli attori (già presentati nel capitolo precedente), con i relativi casi d'uso, ed una definizione del sistema in termini di boundary.

In uscita da questa fase si avranno:

- Il diagramma dei casi d'uso
- Il diagramma delle classi
- I diagrammi di sequenza
- I diagrammi di attività
- Il modello dei dati



4.1 Use case diagram

Descrive le interazioni fra gli attori esterni ed il sistema stesso ciascuna delle quali rappresenta un'azione che un attore può compiere.

Ciascuna funzionalità è offerta dal sistema ed è a disposizione dell'utente che ne presenta un'interazione.

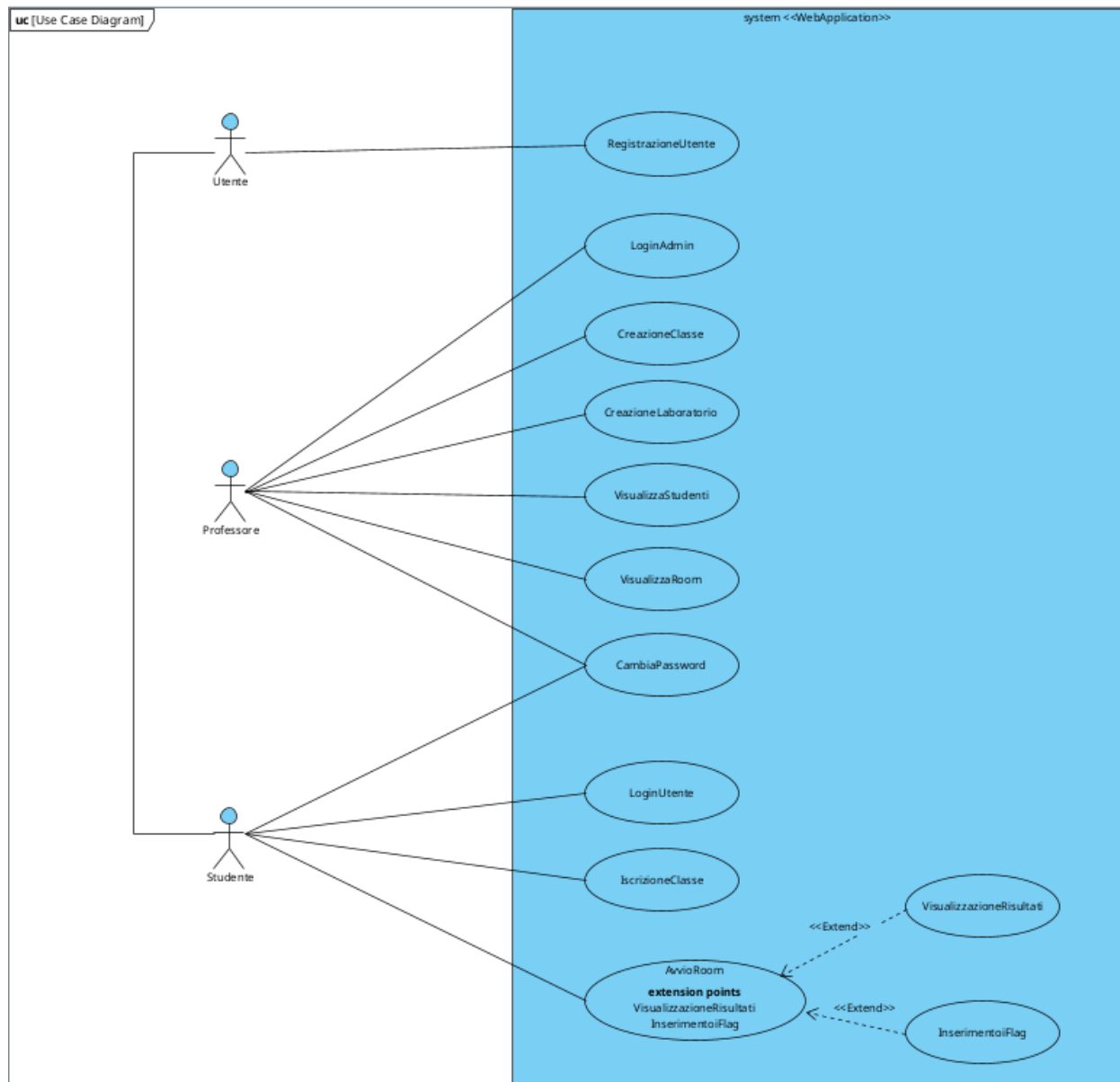


Figura 4.1: Use case diagram

Una cosa interessante da notare è che un utente esterno all'applicazione è abilitato solamente alla registrazione come studente com'è possibile osservare dalla relazione di specificazione.



4.2 Sequence diagram e scenari d'uso

Il sequence diagram è un tipo di diagramma utilizzato nell'ingegneria del software per rappresentare l'interazione tra oggetti all'interno di un sistema software in un determinato scenario d'uso.

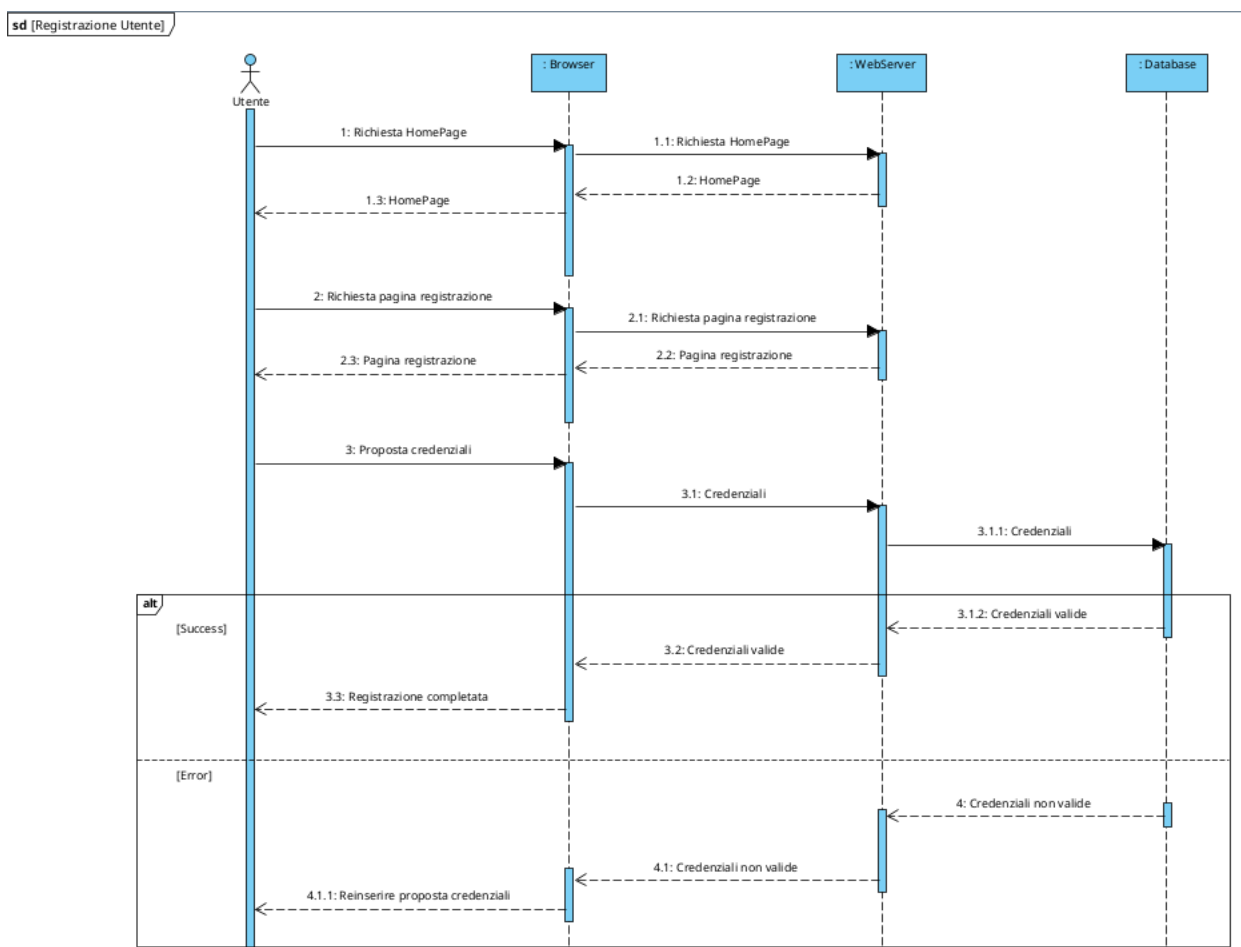
Definiamo scenario d'uso come una descrizione narrativa concreta di come un utente interagisce con un sistema per raggiungere un certo obiettivo; a ciascuno scenario verrà affiancato il relativo sequence diagram che mostra una sequenza di eventi, messaggi e chiamate di metodo tra gli oggetti coinvolti, in un formato di linea temporale.

Il focus di questi diagrammi è una descrizione operativa di un caso d'uso specificando cosa viene fatto da ciascun attore con particolare accento sulla tempificazione delle operazioni; in poche parole è utile a comprendere il flusso di esecuzione del sistema nel tempo.



4.2.1 Registrazione Studente

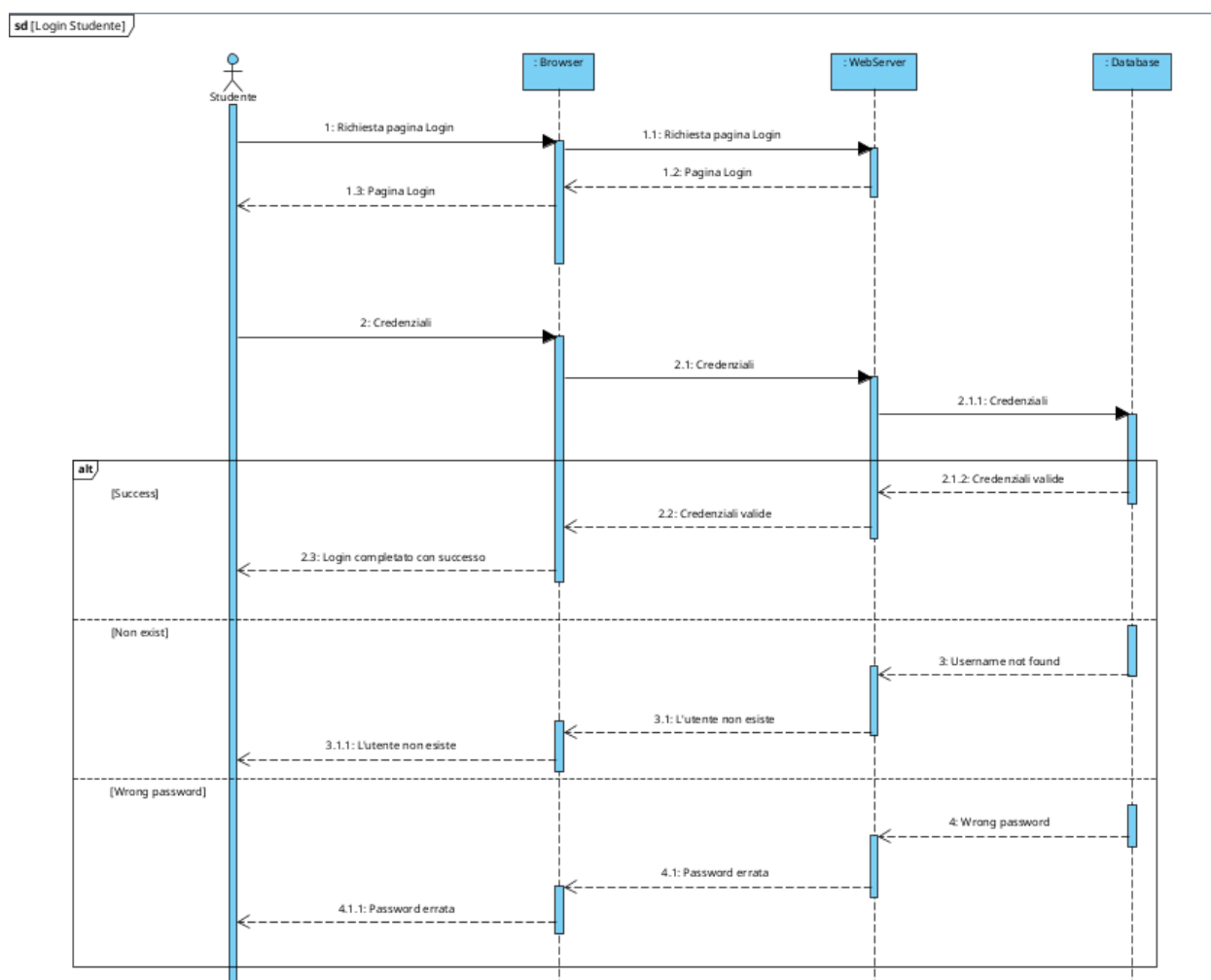
Registrazione Studente	
Portata	Applicazione web di e-learning
Livello	Obiettivo Studente
Attore primario	Studente dell'applicazione
Parti interessate e interessi	Utente (vuole registrarsi come Studente)
Precondizioni	-
Garanzia di successo	Utente viene registrato nella piattaforma e possiede delle credenziali utili ad accedervi
Scenario principale	1. Lo studente si collega alla home page della webapp 2. Lo studente accede alla pagina di registrazione 3. Lo studente inserisce una proposta di credenziali 4. Lo studente viene reindirizzato alla pagina di login
Estensioni	3.a: Credenziali non valide 1. L'applicazione chiede allo studente di inserire nuovamente una proposta di credenziali





4.2.2 Login Studente

Login Studente	
Portata	Applicazione web di e-learning
Livello	Obiettivo Studente
Attore primario	Studente
Parti interessate e interessi	Studente (vuole autenticarsi sulla webapp)
Precondizioni	Lo studente deve avere effettuato la registrazione
Garanzia di successo	Lo studente è abilitato alla navigazione interna del sito
Scenario principale	1. Lo studente accede alla pagina di login 2. Lo studente inserisce le credenziali richieste 3. Lo studente viene reindirizzato alla dashboard
Estensioni	2.a: Lo studente non esiste 1. L'applicazione chiede allo studente di registrarsi o inserire credenziali corrette 2.b: Credenziali errate 1. L'applicazione chiede allo studente di reinserire le credenziali





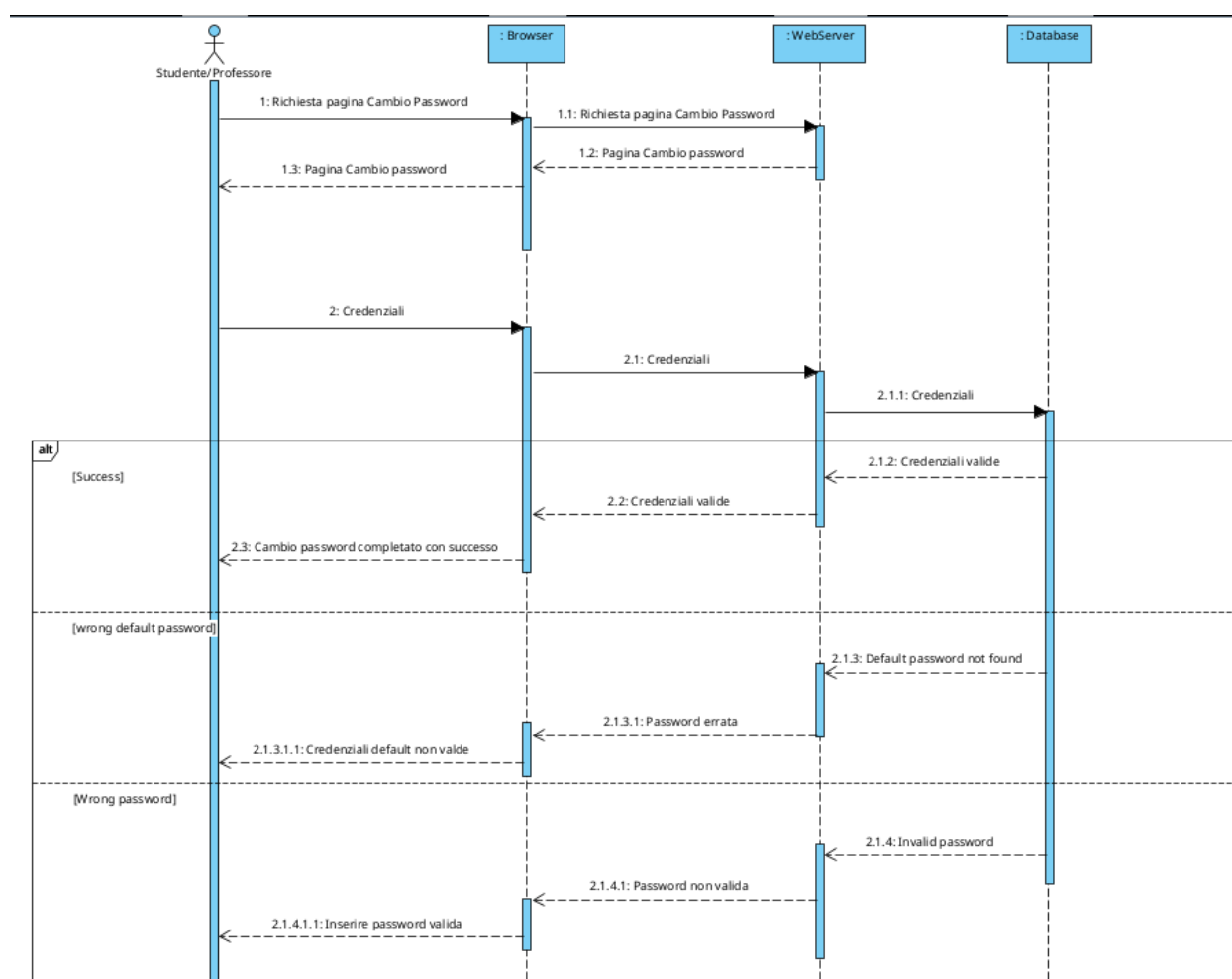
4.2.3 Login Professore

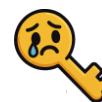
Login Professore		
Portata	Applicazione web di e-learning	
Livello	Obiettivo Professore	
Attore primario	Professore	
Parti interessate e interessi	Professore (vuole autenticarsi sulla webapp)	
Precondizioni	Il professore deve essere in possesso delle credenziali di default	
Garanzia di successo	Il professore è abilitato alla navigazione interna del sito	
Scenario principale	1. Il professore accede alla pagina di login 2. Il professore inserisce le credenziali di default fornite dal reparto IT 3. Il professore viene reindirizzato alla dashboard	
Estensioni	2.a: Il professore non esiste 1. L'applicazione chiede al professore di contattare il reparto IT 2.b: Credenziali errate 1. L'applicazione chiede al professore di reinserire le credenziali	



4.2.4 Cambio Password

Cambio Password	
Portata	Applicazione web di e-learning
Livello	Obiettivo studente e Professore
Attore primario	Studente, Professore
Parti interessate e interessi	Studente, Professore (vogliono modificare la password attuale relativa al proprio account)
Precondizioni	Lo studente/Professore deve essere autenticato
Garanzia di successo	Lo studente/Professore è in grado di accedere alla piattaforma con la nuova password
Scenario principale	1. Lo studente/Professore seleziona la pagina di cambio password 2. Lo studente/Professore inserisce la password attuale e la nuova password 3. Lo studente/Professore completa l'operazione ed attende una conferma dall'applicazione
Estensioni	2.a: Password attuale errata 1. Il sistema chiede allo studente/Professore di inserire nuovamente la password 2.b: Nuova password non valida 1. Il sistema chiede allo studente/Professore di inserire una nuova proposta di password





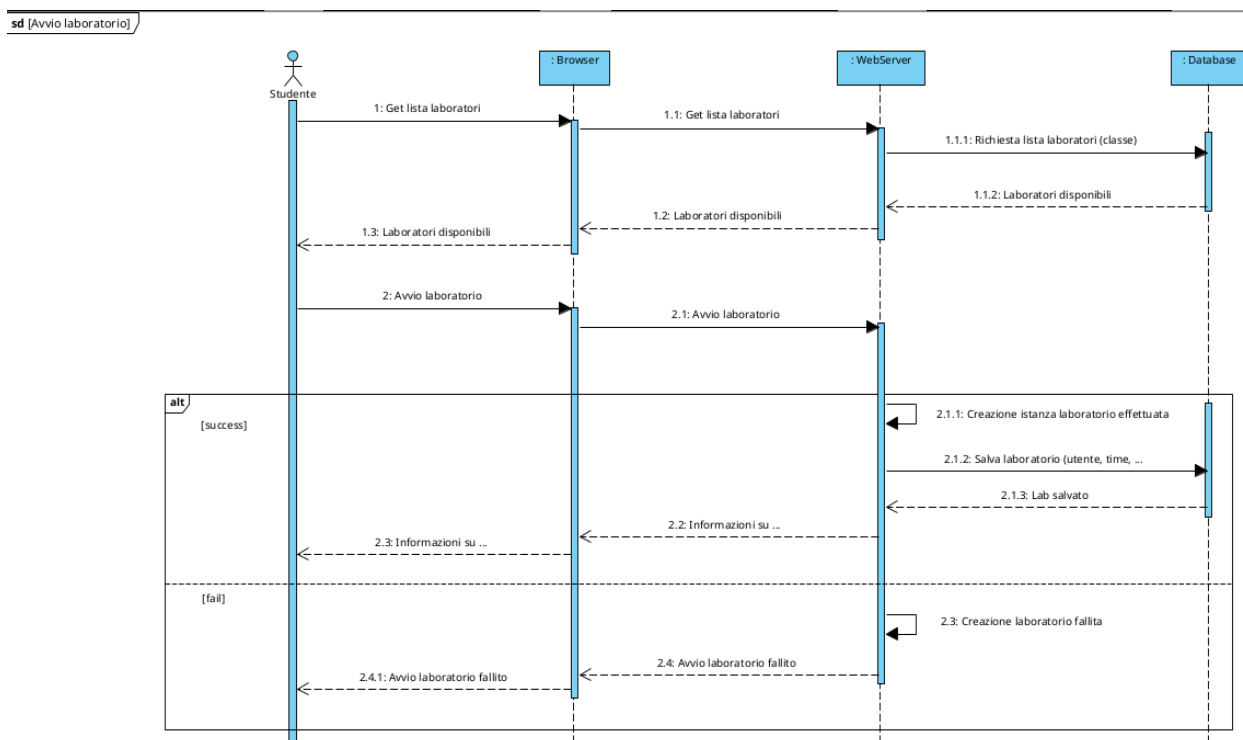
4.2.5 Iscrizione Classe

Iscrizione Classe	
Portata	Applicazione web di e-learning
Livello	Obiettivo Studente
Attore primario	Studente
Parti interessate e interessi	Studente (vuole iscriversi ad una classe fra quelle disponibili)
Precondizioni	Lo studente deve essere autenticato
Garanzia di successo	Lo studente è in grado di effettuare tutte le operazioni relative ai laboratori della classe
Scenario principale	1. Lo studente accede alla sezione di iscrizione delle classi 2. Lo studente invia la richiesta di iscrizione alla classe desiderata fra quelle disponibili 3. Il sistema comunica conferma l'avvenuta registrazione allo studente
Estensioni	3.a: Lo studente è già iscritto alla room 1. Il sistema comunica allo studente che l'iscrizione è stata effettuata precedentemente



4.2.6 Avvio Room

Avvio Room	
Portata	Applicazione web di e-learning
Livello	Obiettivo Studente
Attore primario	Studente
Parti interessate e interessi	Studente (vuole avviare un laboratorio)
Precondizioni	1. Lo studente deve essere autenticato 2. Lo studente deve essere iscritto ad una classe 3. Il professore della classe in questione deve aver aggiunto almeno una room
Garanzia di successo	Il sistema avvia una room e fornisce un meccanismo di collegamento allo studente
Scenario principale	1. Lo studente naviga la lista delle room disponibili da un'apposita sezione 2. Lo studente manda la richiesta di start della room 3. Il sistema fornisce dettagli e comando per il collegamento alla room 4. Il sistema salva le informazioni relative alla room
Estensioni	3.a: Il sistema non riesce ad avviare il laboratorio 1. Il sistema comunica allo studente di riprovare più tardi

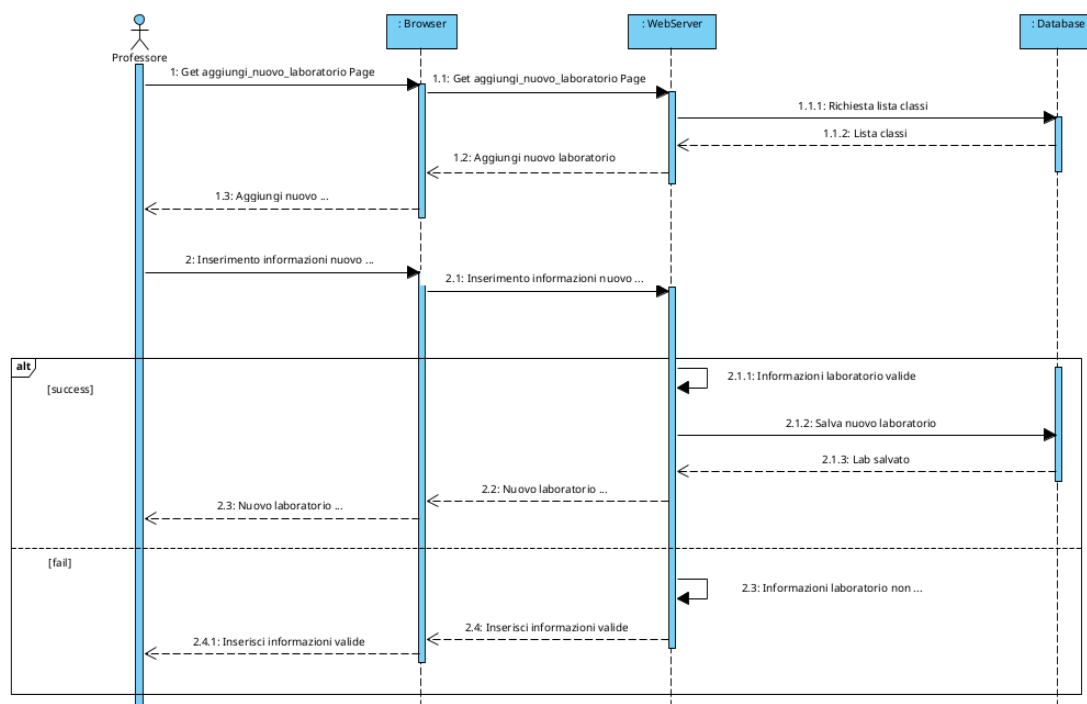




4.2.7 Creazione Room

Creazione Laboratorio	
Portata	Applicazione web di e-learning
Livello	Obiettivo Professore
Attore primario	Professore
Parti interessate e interessi	Professore (vuole aggiungere una room ad una classe)
Precondizioni	1. Il professore deve essere autenticato 2. Il sistema deve possedere almeno una classe
Garanzia di successo	Nella sezione delle room disponibili relativi alla classe sarà presente la room aggiunta
Scenario principale	1. Il professore accede alla schermata di aggiunta della room 2. Il professore inserisce le informazioni relative alla room (testuali e file.yaml) 3. Il professore invia le informazioni alla piattaforma 4. Il sistema salva le informazioni del laboratorio 5. Il sistema risponde con un messaggio di conferma
Estensioni	2.a: Tipo di file non coerente 1. Il sistema chiede al professore di controllare l'estensione del file caricato 2.b: Nome della room già utilizzato 1. Il sistema chiede al professore di inserire una nuova proposta per il nome della room

sd [Creazione laboratorio]





4.2.8 Creazione Classe

Creazione Classe	
Portata	Applicazione web di e-learning
Livello	Obiettivo Professore
Attore primario	Professore
Parti interessate e interessi	Professore (vuole creare una nuova classe)
Precondizioni	Il professore deve essere autenticato
Garanzia di successo	Gli utenti troveranno una nuova classe nella lista delle classi disponibili all'iscrizione
Scenario principale	1. Il professore accede alla schermata di aggiunta della classe 2. Il professore inserisce le informazioni relative alla classe (testuali) 3. Il professore invia le informazioni alla piattaforma 4. Il sistema risponde con un messaggio di conferma
Estensioni	2.a: Classe già esistente 1. Il sistema chiede al professore di inserire una nuova proposta per il nome della classe

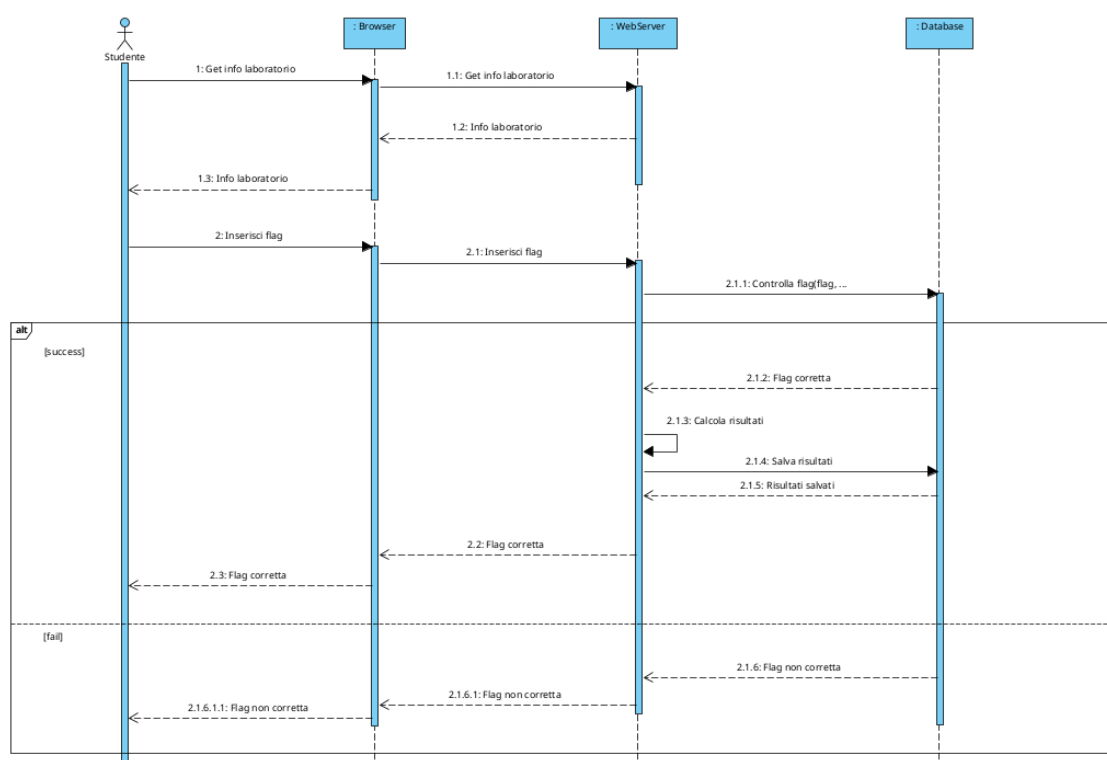
4.2.9 Visualizza Risultati

Visualizza Risultati	
Portata	Applicazione web di e-learning
Livello	Obiettivo Professore, obiettivo Studente
Attore primario	Professore, Studente
Parti interessate e interessi	Professore, Studente (vuole visualizzare i risultati relativi ad una room)
Precondizioni	1. Il professore/studente deve essere autenticato 2. Deve esistere almeno una room 3. Almeno uno studente deve aver completato la room
Garanzia di successo	Lo studente/professore consulta i punteggi relativi alla room selezionata
Scenario principale	1. Il professore/studente accede alla schermata di visualizzazione dei risultati 2. Il professore/studente seleziona la classe e la room 3. Il sistema genera la sezione dei risultati
Estensioni	-



4.2.10 Inserimento Flag

Inserimento Flag	
Portata	Applicazione web di e-learning
Livello	Obiettivo studente
Attore primario	Studente
Parti interessate e interessi	Studente (vuole dimostrare di aver completato il laboratorio)
Precondizioni	1. Lo studente deve essere autenticato 2. Lo studente deve iscritto ad una classe 3. Lo studente deve aver avviato un laboratorio 4. Lo studente deve essere in possesso della flag relativa al laboratorio
Garanzia di successo	Lo studente ha completato il laboratorio (ha prodotto un risultato che può essere pubblicato)
Scenario principale	1. Lo studente accede alla schermata di informazioni del laboratorio avviato 2. Lo studente inserisce la flag testuale all'interno dell'apposito campo 3. Lo studente invia la flag al sistema 4. Il sistema conferma l'inserimento della flag e chiude il laboratorio
Estensioni	4.a: Flag errata 1. Il sistema comunica allo studente di aver inserito una flag errata





4.2.11 Visualizzazione Studenti

Visualizzazione Risultati	
Portata	Applicazione web di e-learning
Livello	Obiettivo Professore
Attore primario	Professore
Parti interessate e interessi	Professore (vuole consultare la lista degli studenti iscritti ad una classe)
Precondizioni	1. Il professore deve essere autenticato
Garanzia di successo	Il professore visualizza la lista degli studenti relativi alla classe selezionata
Scenario principale	1. Il professore accede alla schermata di visualizzazione degli studenti 2. Il professore seleziona una classe dall'elenco di quelle disponibili 3. Il sistema produce la lista degli studenti iscritti a quella classe
Estensioni	-

4.2.12 Visualizzazione Room

Visualizzazione Room	
Portata	Applicazione web di e-learning
Livello	Obiettivo Professore
Attore primario	Professore
Parti interessate e interessi	Professore (vuole consultare la lista delle room associate ad una classe)
Precondizioni	1. Il professore deve essere autenticato
Garanzia di successo	Il professore visualizza la lista delle room relative alla classe selezionata
Scenario principale	1. Il professore accede alla schermata di visualizzazione delle room 2. Il professore seleziona una classe dall'elenco di quelle disponibili 3. Il sistema produce la lista delle room relative alla classe selezionata
Estensioni	-



4.3 Activity Diagram

Questa tipologia di diagramma viene utilizzata per descrivere il flusso di lavoro o di attività all'interno di un sistema software o di un processo di business.

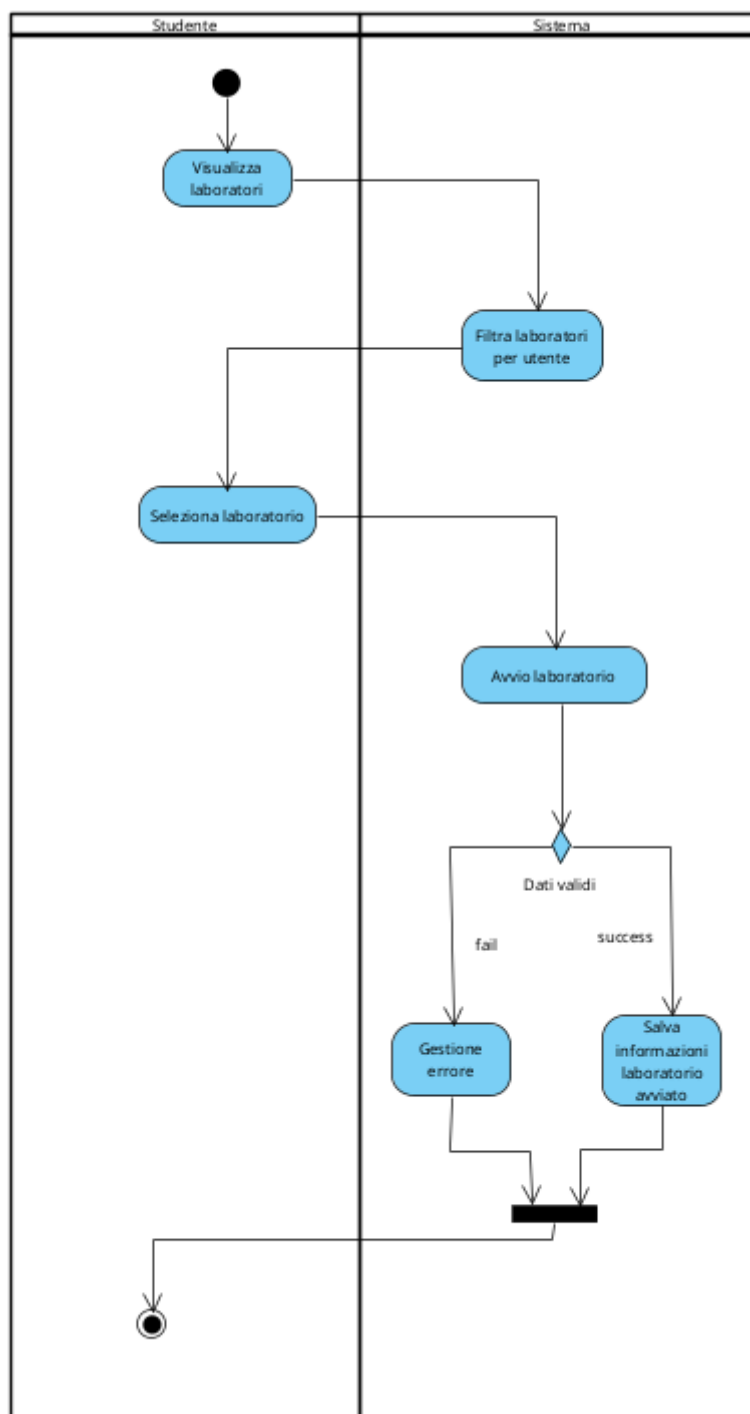
Utile per descrivere diversi tipi di processi, come ad esempio processi di business, processi di produzione o processi software.

A differenza del diagramma di sequenza, permette di seguire il flusso di attività interattiva fra attori e sistema di ciascun caso d'uso; essendo attività molti simili fra loro, sono stati realizzati solamente i diagrammi più utili per comprendere le tipologie di attività che vengono compiute dal sistema.

In tutti i successivi diagrammi i due attori che dialogano sono semplificati in “*chi richiede il servizio*” ed il “*sistema*” in quanto non siamo interessati alla scomposizione di quest'ultimo ma semplicemente a quale attività viene effettuata.

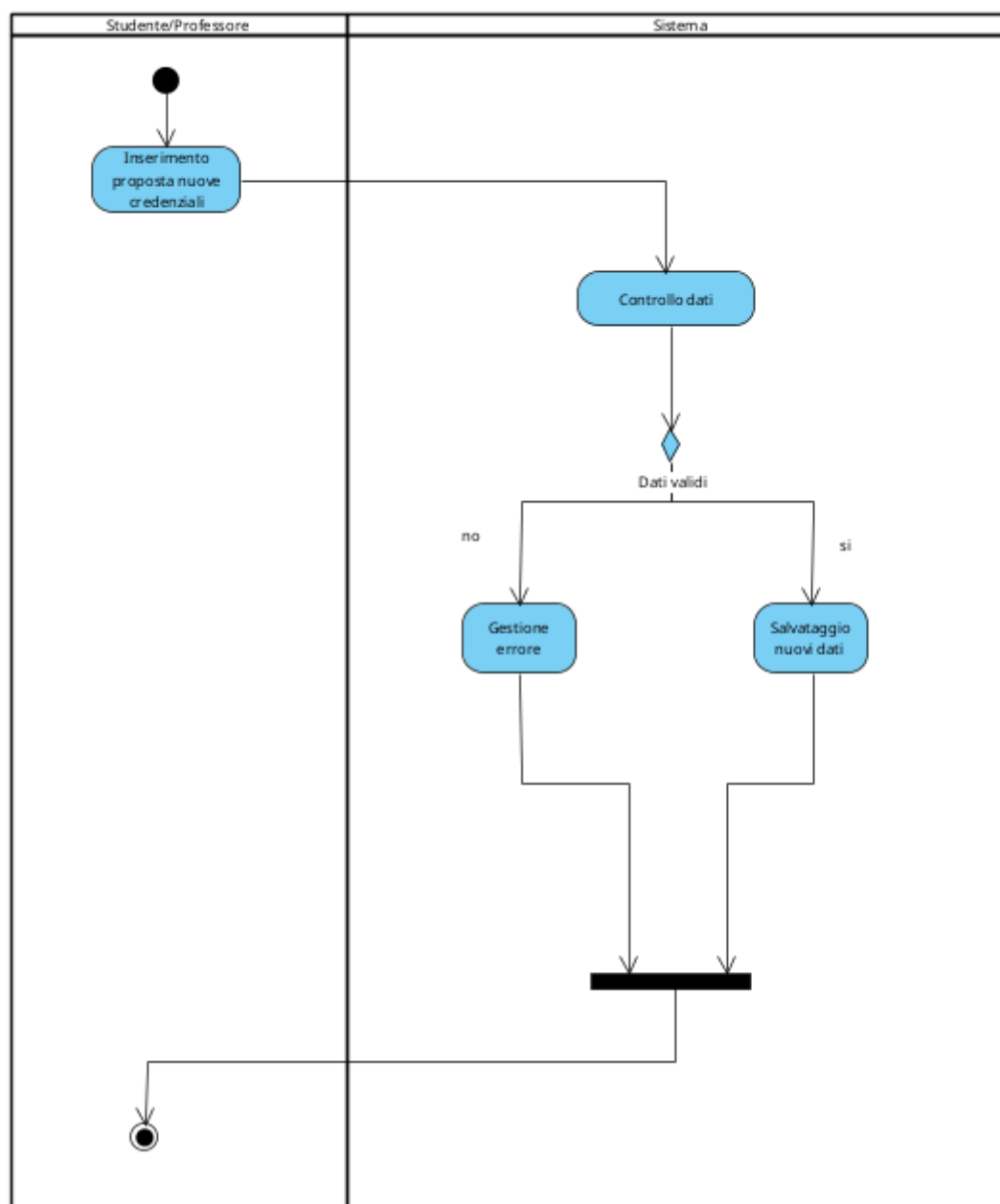


4.3.1 Avvio Laboratorio



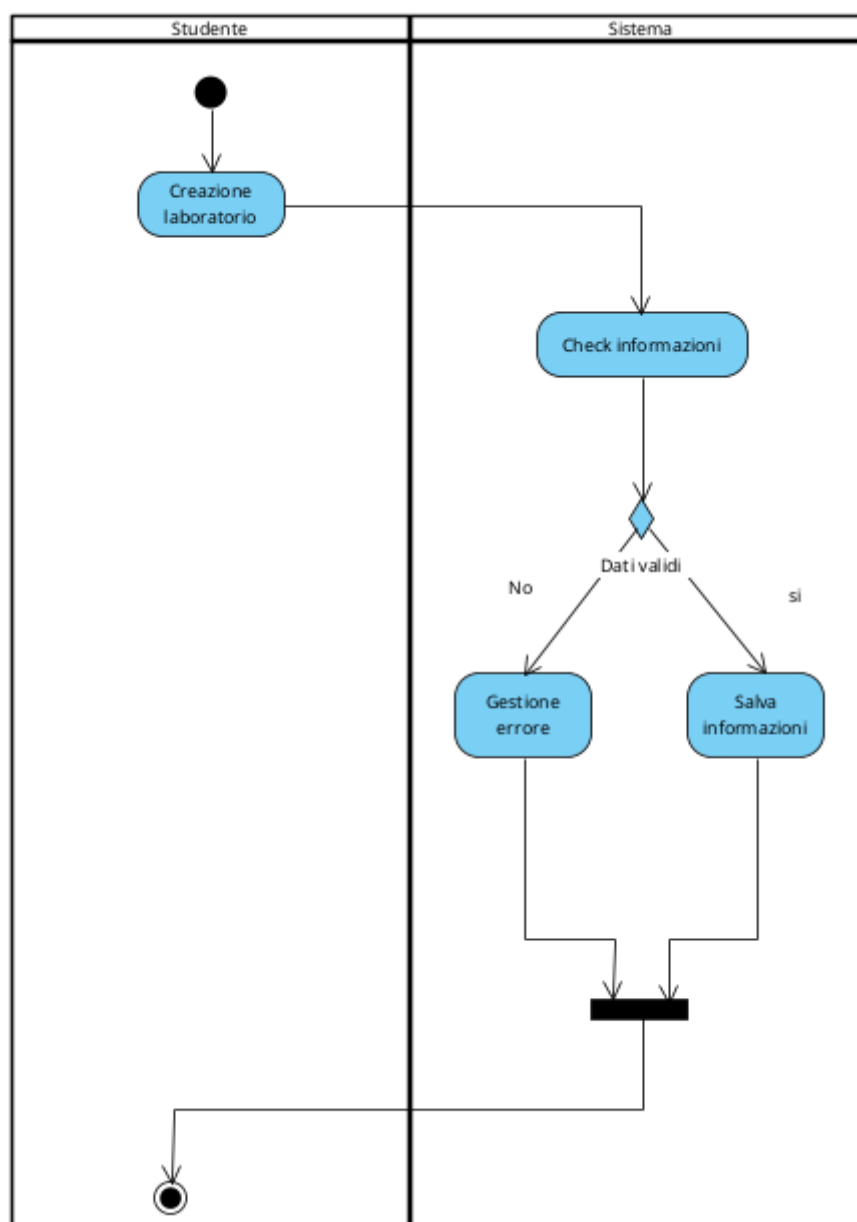


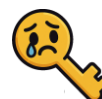
4.3.2 Cambio Password



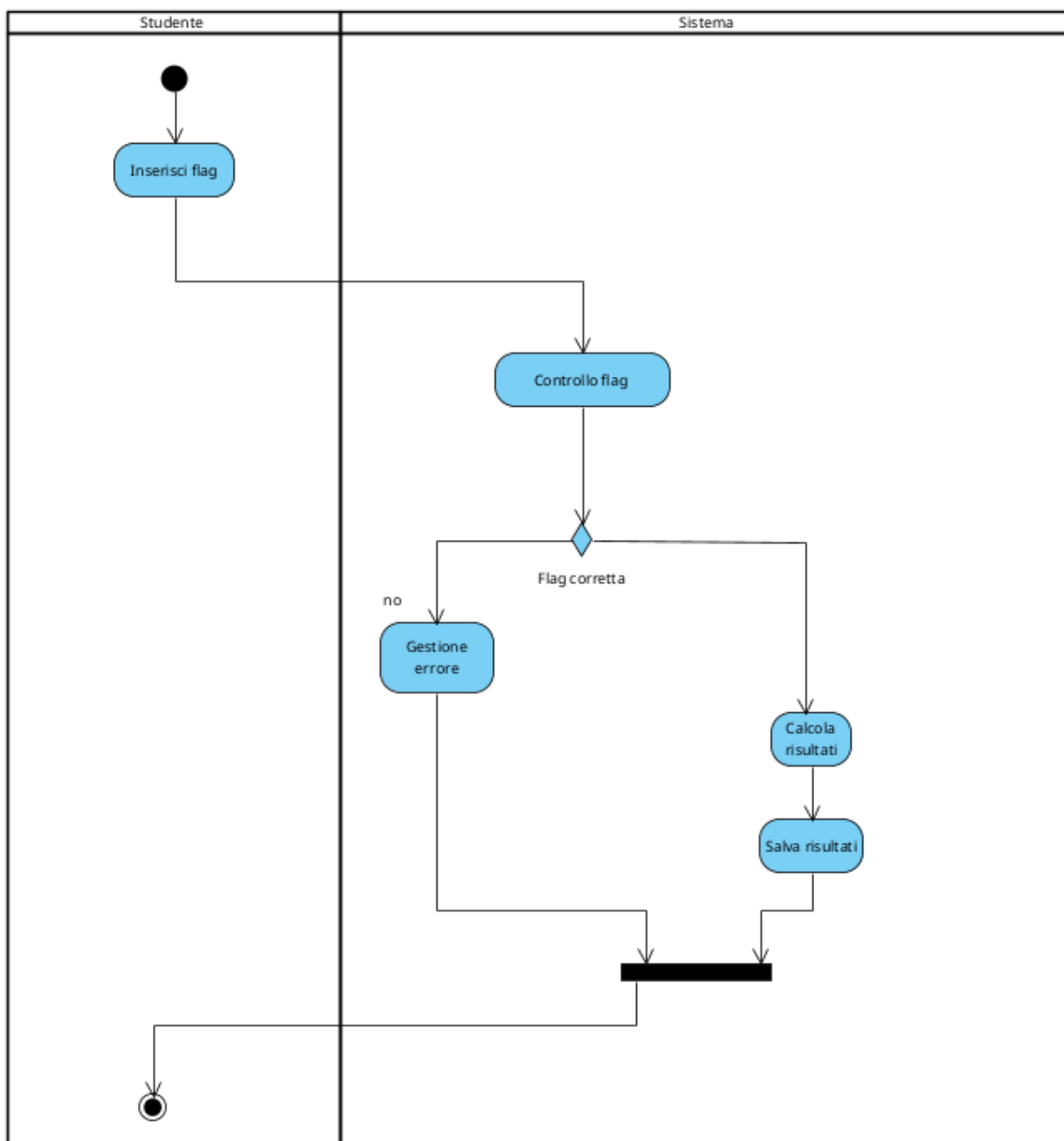


4.3.3 Crea Laboratorio





4.3.4 Inserimento Flag





4.4 Class diagram

Un Class Diagram è un diagramma UML che rappresenta le classi di un sistema, i loro attributi, metodi e le relazioni tra loro

Rappresenta uno strumento utile a modellare la struttura statica del software dividendo il sistema in oggetti e classi; è uno dei diagrammi che molto spesso viene raffinato nelle varie fasi dello sviluppo assottigliando sempre di più la distanza fra l'analisi teorica e l'implementazione pratica.

Grazie ad esso è possibile chiarire le relazioni, le responsabilità e le operazioni di ciascuna classe già nella fase di analisi e senza andare ad effettuare nessun tipo di specifica riguardo la loro implementazione.

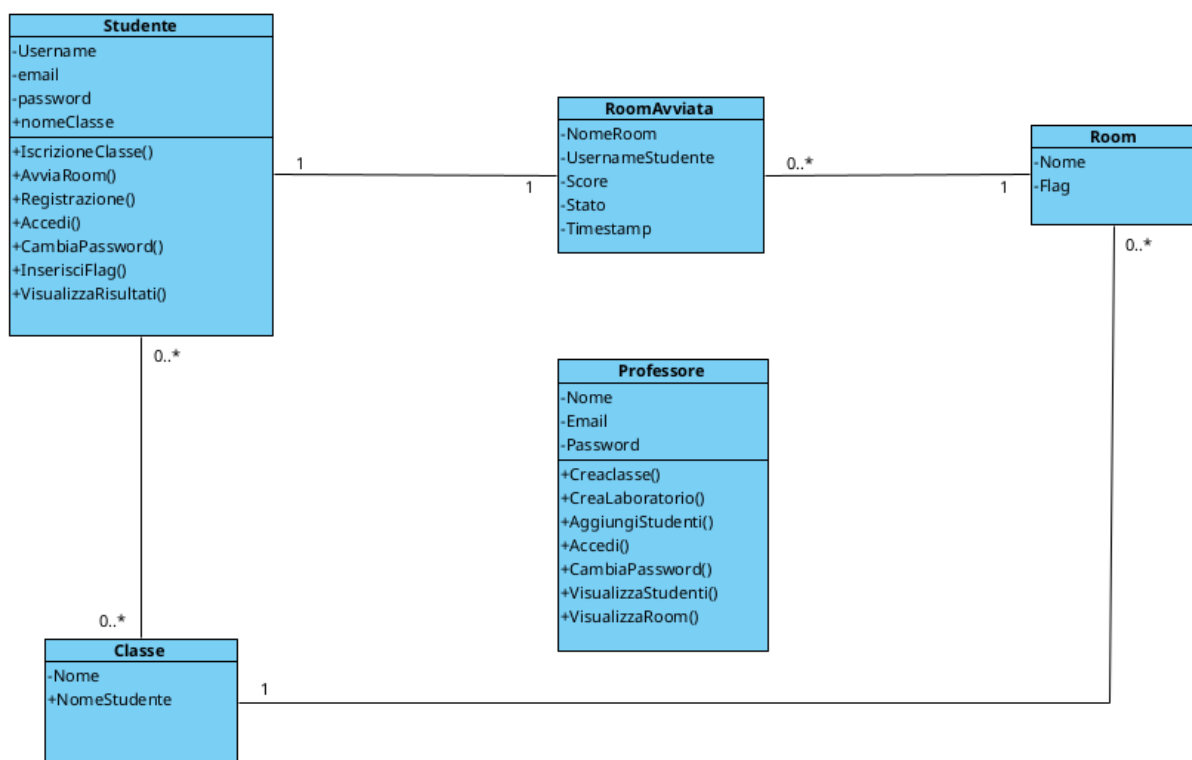


Figura 4.2: Class diagram

Nel grafico presentato è possibile notare come non ci siano, a questo punto dell'analisi, alcuna informazione circa le specifiche degli attributi delle interazioni che verranno presentate nei successivi diagrammi.



4.5 Entity Relation Diagram

Un Entity-Relationship Diagram è un modello visuale che rappresenta la struttura logica di un database ed è spesso utilizzato per descrivere le entità principali del sistema, i loro attributi e le relazioni tra di esse.

Gli ER diagram sono uno strumento fondamentale nella fase di progettazione del database, perché aiutano a tradurre i concetti del dominio applicativo in una struttura dati facilmente implementabile.

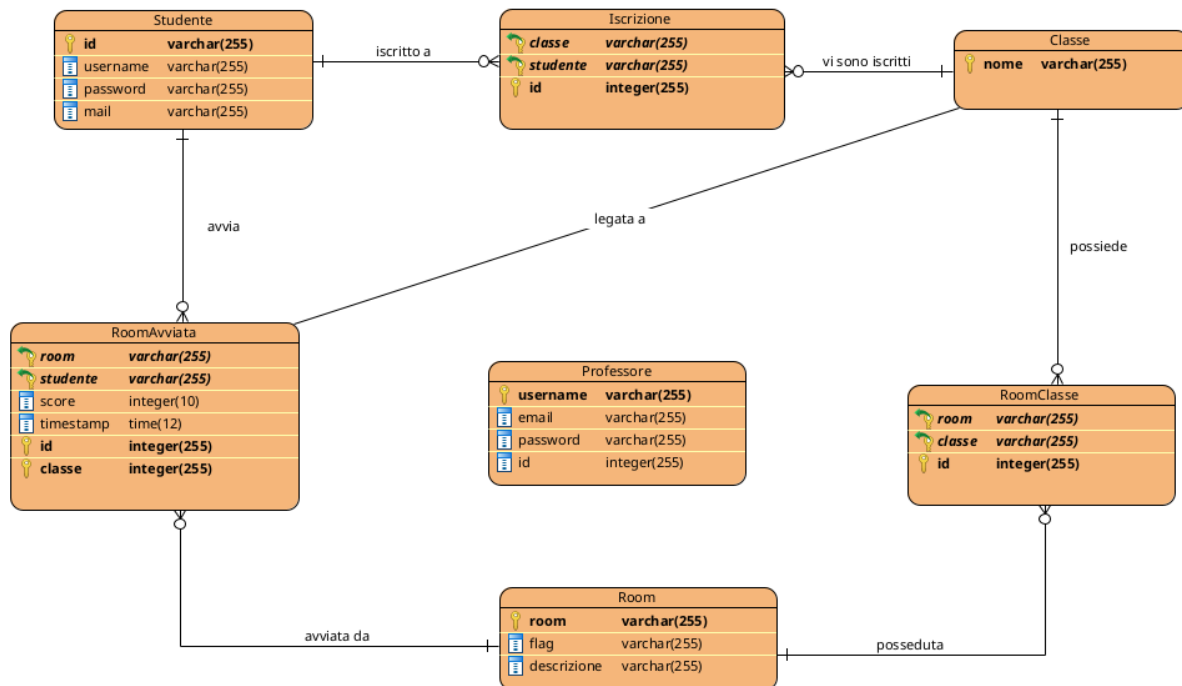


Figura 4.3: Entity Relation Diagram

In esso possiamo notare come, a partire anche dal brainstorming fatto sul diagramma delle classe, sia molto semplice pensare di realizzare un database a partire da un modello che ne rappresenta una sua astrazione.



5. FASE DI PROGETTAZIONE

Il passo successivo alla fase di analisi è la fase di progettazione durante la quale si tenta di realizzare i requisiti e le strutture identificate nella fase precedente attraverso un insieme di componenti software legati dalla scelta architetturale che viene fatta in questi momenti.

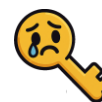
Le scelte che sono state fatte sono le seguenti:

- **Client-server** come stile architetturale grazie al quale si è in grado di garantire la comunicazione fra studenti/professori e sistema attraverso l'uso di strumenti comuni.
- **Model-view-control** come pattern architetturale con cui si è riusciti a separare logicamente le funzioni di gestione delle entità, la business logic e la presentazione dei dati agli utenti.

Questa fase rappresenta il collegamento fondamentale tra l'analisi e l'implementazione, ed è cruciale per garantire la qualità e la manutenibilità del sistema finale.

Nel seguito di questa sezione verranno presentati:

- Component diagram
- Package diagram
- Deployment diagram



5.1 Component diagram

Il Component Diagram è un utile strumento per descrivere l'architettura di un sistema software e le interazioni tra i suoi componenti. Analizzando il Component Diagram di un sistema, è possibile avere una visione ad alto livello della struttura del sistema e delle relazioni tra i suoi componenti. In particolare, i componenti principali del sistema e le loro interfacce, nonché le dipendenze tra di essi, possono essere esplorati e valutati.

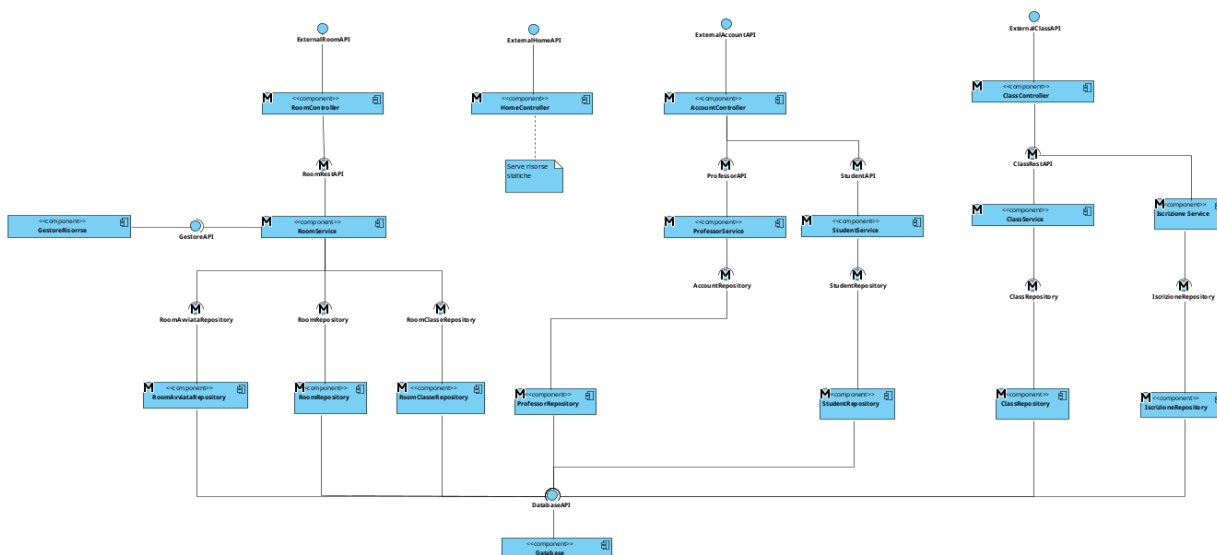


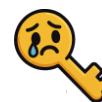
Figura 5.1: Component diagram

Da qui si può notare come la scelta di adottare un pattern MVC sia evidentemente concretizzata nella delega delle funzioni fra i vari component:

- **Controller** che si occupano di fornire le API con cui gli utenti possono contattare e richiedere le varie operazioni. Sono stati organizzati in modo tale da non generare confusione logica fra i compiti assegnati a ciascuno di essi; in questo modo risulta molto semplice sia modularizzare i vari compiti che comprendere i flussi delle varie operazioni (semplificato anche dall'aver scelto come controller le entità definite nel entity diagram).
- **Model** concretizzati attraverso le repository che si occupano di garantire la persistenza dei dati attraverso le chiamate ad un component DB.
- **Service** che rappresentano la business logic legata a ciascun model e che vengono invocati direttamente dai controller che dirottano le rotte.
- **View** distribuite dai vari controller (principalmente HomeController) sotto forma di pagine statiche.

Un focus particolare va fatto sul component Gestore_Risorse che viene contattato attraverso delle API dedicate solamente dal roomService in quanto ne rappresenta un tentativo di semplificazione della logica di lavoro legata alle room; questa scelta permette sia di separare per tipologia le operazioni sulle room sia di scegliere, successivamente, una tecnologia per l'implementazione delle istanze dei laboratori che sia più ottimizzata e focalizzata a quello specifico obiettivo.





5.2 Package diagram

Il package diagram è un modello UML utile per mostrare come i vari pacchetti (package) in una applicazione software sono organizzati e comunicano tra loro. In sostanza mostra la struttura ad alto livello del sistema, in modo da aiutare a comprendere la modularizzazione delle componenti.

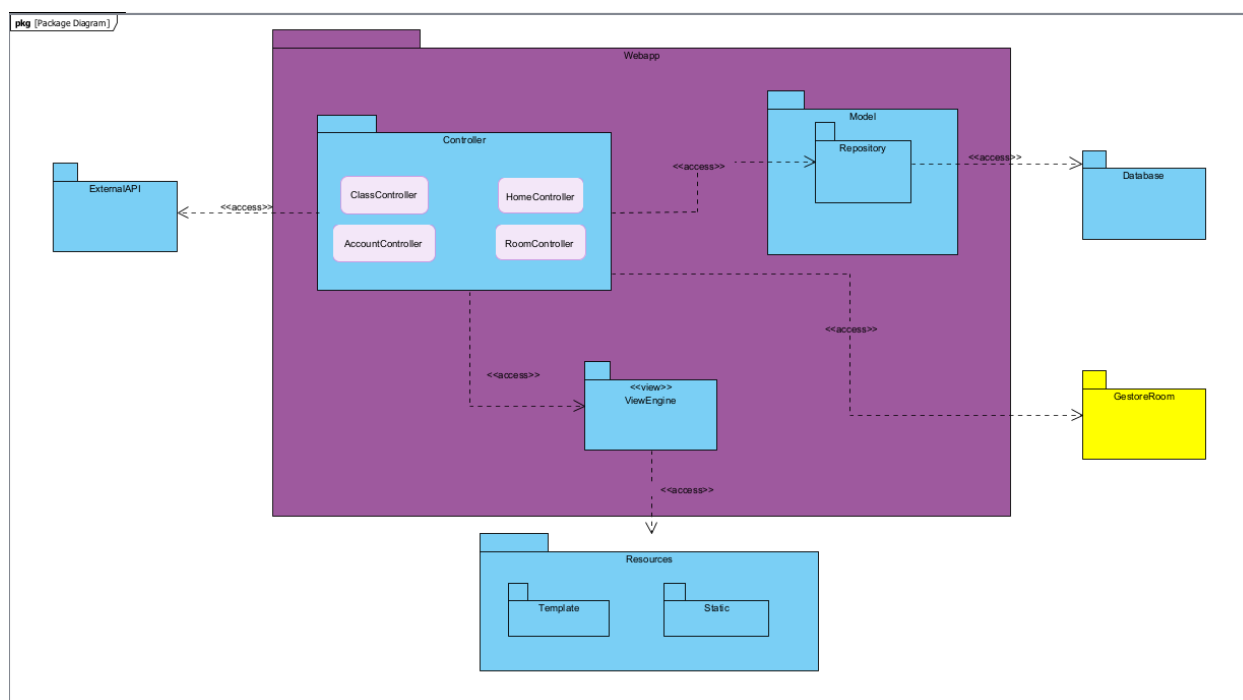


Figura 5.2: Package Diagram

L'immagine mostra come la nostra applicazione sia strutturata, evidenziando l'organizzazione dei vari blocchi e le loro interazioni. In particolare, viene messa in risalto la centralità del pacchetto Webapp, che racchiude la maggior parte della logica di funzionamento. Da notare come si è preferito organizzare il lavoro del controller in istanze differenti in modo tale da dividere il lavoro in funzione del dominio esecutivo aumentando così la coesione dei vari componenti.

Al suo interno, i componenti principali della Webapp (Controller, Model e View) comunicano con elementi esterni come Database e gestore delle risorse. È importante notare come il lavoro dei controller sia stato suddiviso in istanze distinte con l'obiettivo di separare le responsabilità in base al dominio applicativo, migliorando così la coesione e facilitando la manutenzione del sistema.

Questo diagramma è particolarmente utile perché fornisce una visione d'insieme dell'architettura prevista, permettendo di identificare sin da subito i principali attori coinvolti e il ruolo di ciascun componente nel corretto funzionamento dell'applicazione.



5.3 Deployment diagram

Il diagramma di deployment descrive la distribuzione fisica dei componenti software del sistema sui diversi nodi dell'infrastruttura.

Dopo aver preso tutte le decisioni progettuali del caso, è possibile mostrare anche dove i vari componenti dell'applicazione saranno infine allocati andando quindi anche a specificare le caratteristiche delle macchine che ospiteranno i vari moduli

Questo diagramma permette di visualizzare dove vengono eseguiti i vari moduli applicativi, come sono collegati tra loro e quali risorse hardware o virtuali intervengono nell'esecuzione del sistema.

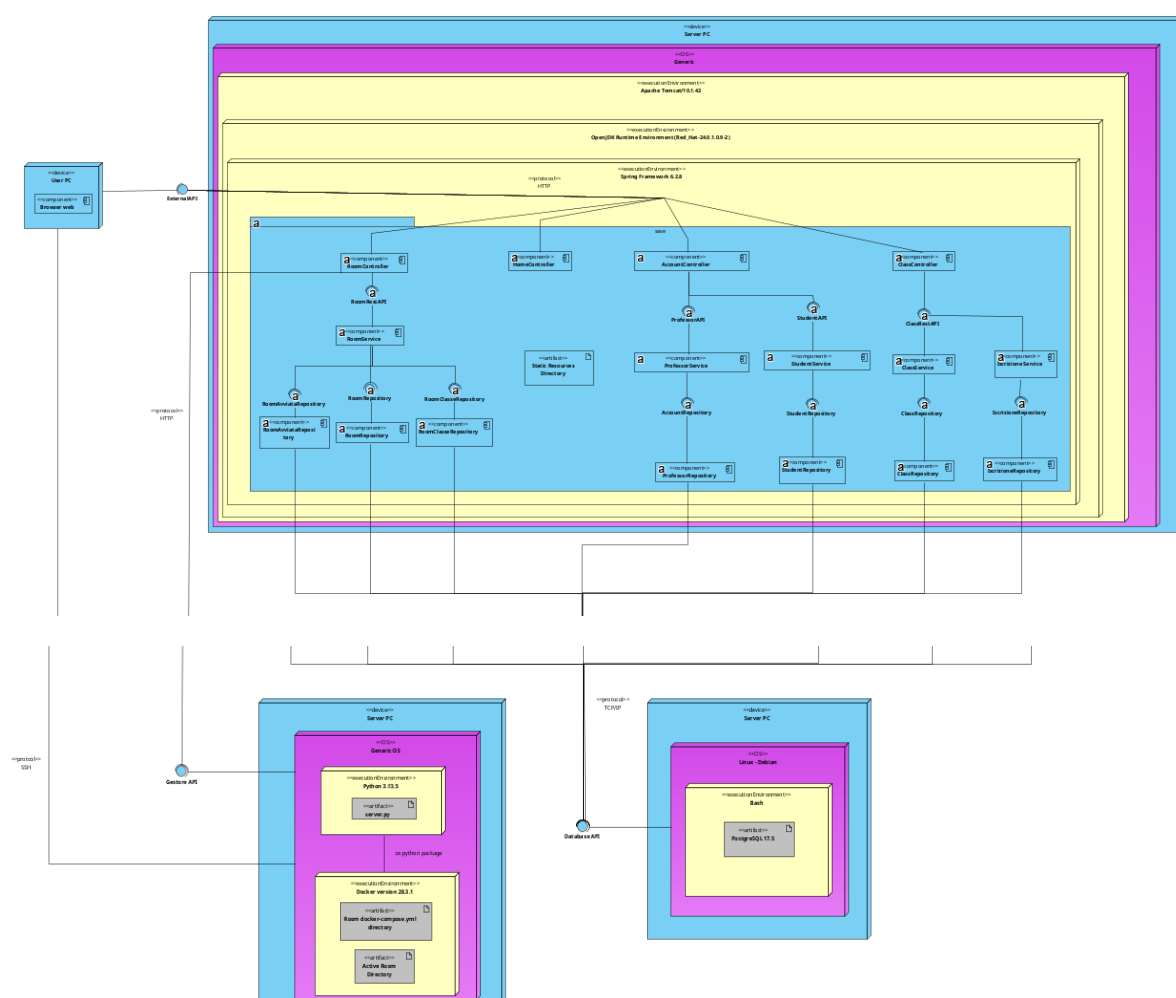


Figura 5.3: Deployment diagram



È importante sottolineare che in questa fase ci si occupa anche di preparare gli ambienti di esecuzione per cui è fondamentale scegliere bene e tenere conto di tutte le dipendenze per assicurare un funzionamento.

In questo caso possiamo osservare come ci siano anche delle specifiche relative ai protocolli con i quali vengono contattati i vari nodi dell'applicazione ed i dettagli circa le versioni dei container utilizzati (docker e jvm).

Per il diagramma relativo al deploy dell'applicazione completamente containerizzato si rimanda ai capitoli successivi.



6. IMPLEMENTAZIONE

In questo capitolo vengono presentati i vari sequence di dettaglio implementati scegliendo tra le varie funzionalità offerte dall'applicazione.

6.1 Sequence diagram di dettaglio

I sequence di dettaglio sono un tipo di diagramma dinamico che forniscono una visione dettagliata dell'interazione tra gli oggetti all'interno di un sistema software. Questi diagrammi mostrano le chiamate di metodo tra gli oggetti nel corso del tempo e illustrano l'ordine e il flusso delle operazioni. Nei paragrafi successivi vengono presentati i sequence diagram delle due funzionalità principali dell'applicazione software.

6.1.1 Crea laboratorio

Il professore si trova nella sezione della webapp di creazione di un nuovo laboratorio ed inserisce una proposta di nome della room, la quale azione scatena l'attivazione del RoomController (RoomPresente). Questo controller chiama un service per verificare se quel nome inserito è già presente nel database informando di conseguenza il controller e successivamente l'utente.

Se la proposta di nome room va a buon fine il professore continua ad inserire le informazioni mancanti del nuovo laboratorio e, sempre attraverso la stessa sezione della pagina web, scatena l'attivazione di un altro metodo del controller che a sua volta invoca il service *salvanuovaRoom()* che, mediante un metodo del *roomRepository*, controlla se l'associazione tra room e classe è già presente.

Se il repository comunica al service che l'associazione non è presente nel database, il service invoca un metodo per salvare le informazioni del nuovo laboratorio anche all'interno del gestore delle risorse e, se tutto va a buon fine, il service effettua una chiamata finale al repository per salvare la nuova associazione; infine viene consegnato al controller un messaggio che lo informa sull'esito delle operazioni in modo da presentare i risultati all'utente.

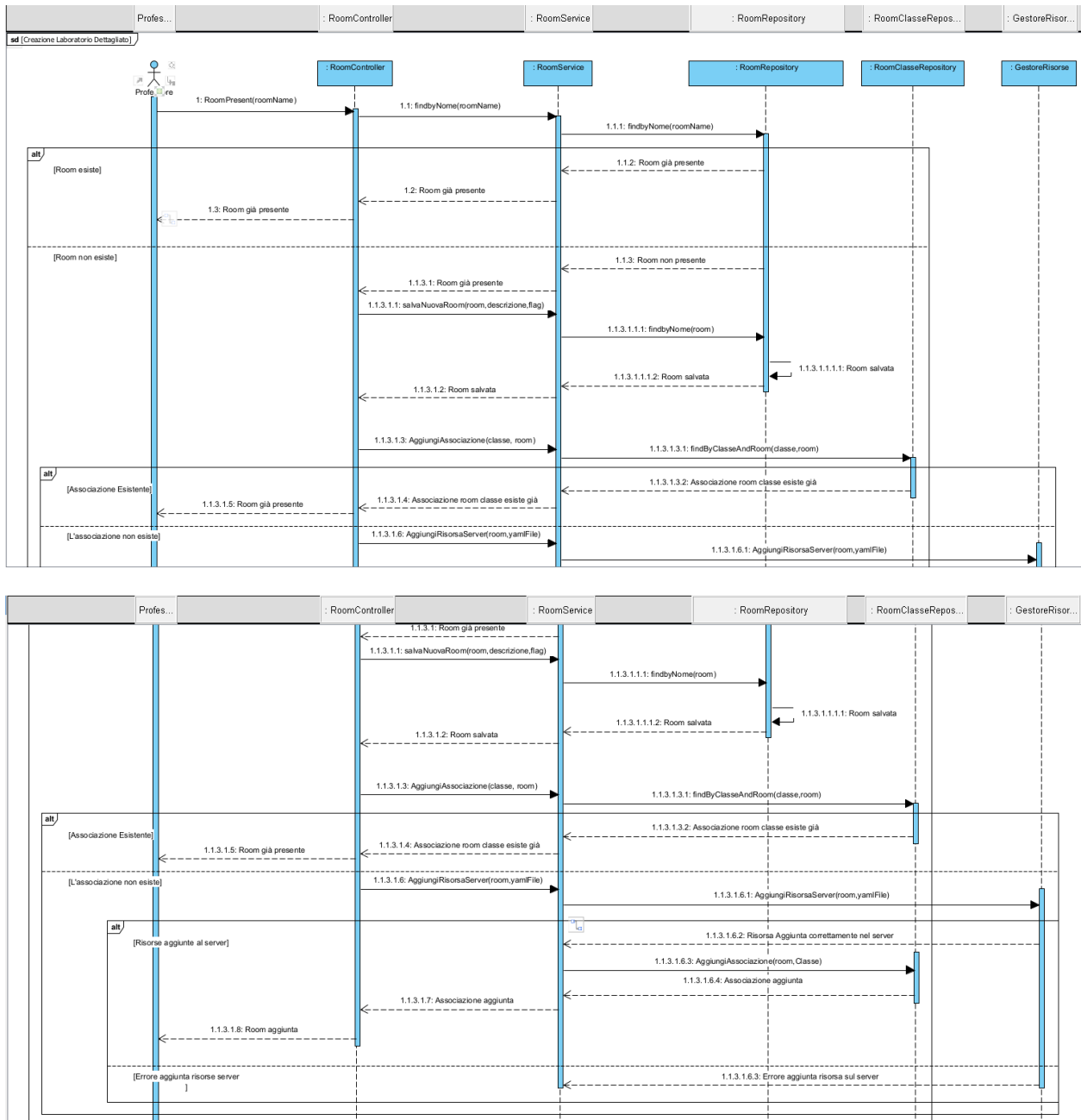


Figura 6.1: Sequence di dettaglio Crea Laboratorio



6.1.2 Avvia laboratorio

L'utente clicca su una delle classi a cui è iscritto e questo scatena l'attivazione del RoomController, il quale attraverso un Roomservice, contatta il RoomClasseRepository per ottenere la lista di laboratori appartenenti a quella classe.

Una volta ottenuta la lista di laboratori l'utente ne sceglie uno e ciò scatena l'attivazione del RoomController il quale invoca il RoomService al fine di verificare se l'associazione tra utente e room è già presente nel roomAvviataRepository.

Se questa associazione è presente il RoomService contatta il gestore delle risorse per avviare direttamente la room e si mette in attesa di una risposta da parte sua per poi inoltrarla al RoomController.

Se l'associazione non è presente il RoomService contatta prima il gestore delle risorse in modo da avviare il laboratorio, e, se il gestore restituisce un messaggio di avvio corretto, contatta nuovamente il RoomRepository per salvare la nuova associazione nella RoomAvviataRepository;

Una volta ultimate queste operazioni in ogni caso il RoomService inoltra l'esito al controller in modo da presentare i risultati allo studente.

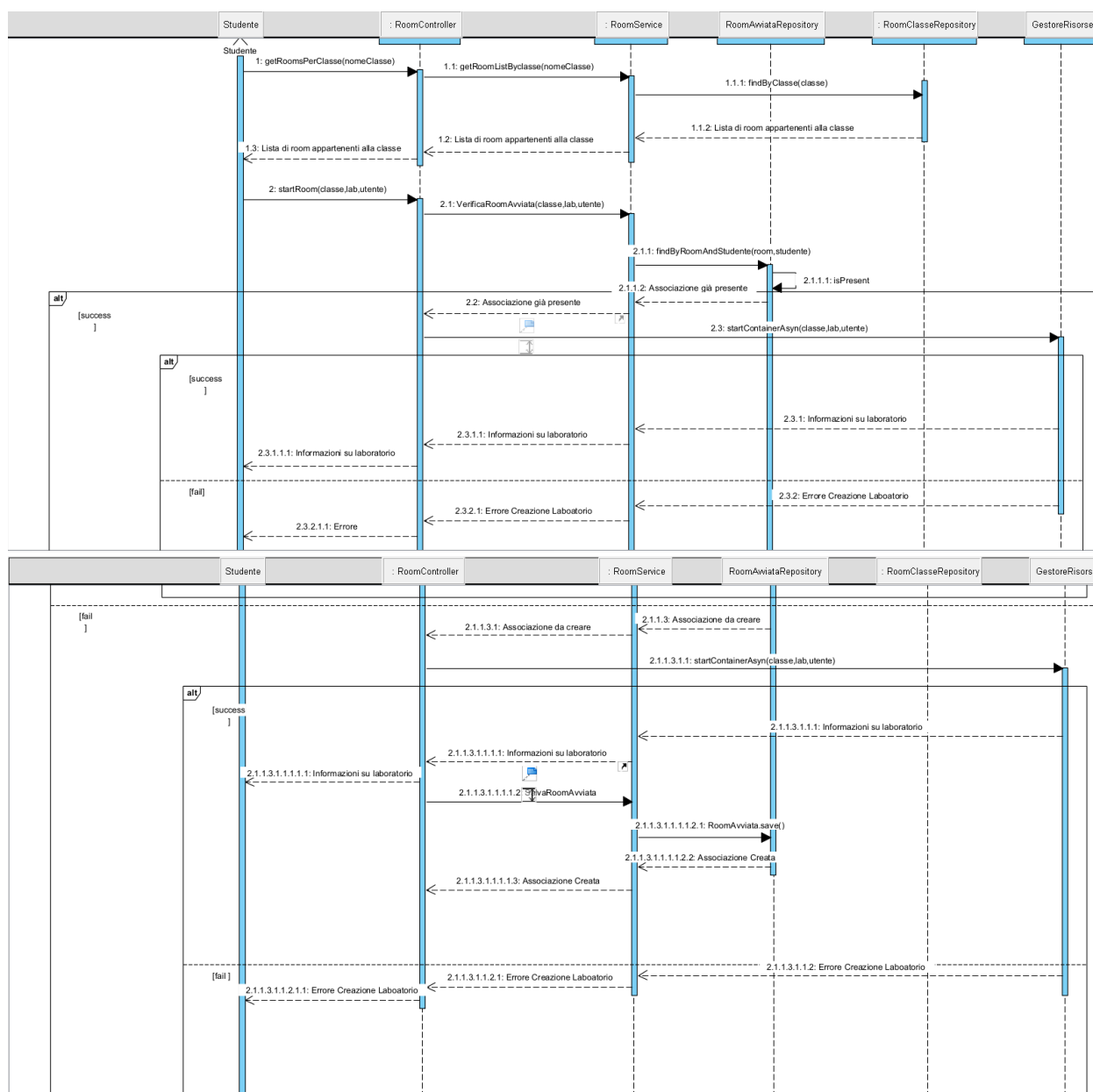


Figura 6.2: Sequence di dettaglio Avvia Laboratorio



6.2 Interfacce

Per quanto riguarda la documentazione delle interfacce utilizzate è stato utilizzato uno strumento (javadoc) di documentazione incluso nel JDK che permette di generare documentazione HTML leggibile a partire da commenti speciali inseriti nel codice java.

Questo strumento ci ha permesso di facilitare la documentazione di classi, metodi ed attributi snellendo di molto il lavoro di documentazione delle API.

Per la generazione della documentazione delle API è stato necessario specificare l'utilizzo di questo strumento all'interno dei plugin ed etichettare tutte le classi seguendo il formato mostrato in figura:

```
/**
 * Gestione delle richieste di cambio della password.
 *
 * @param newPassword nuova password scelta
 * @param oldPassword password attuale
 * @param request      oggetto HTTP request
 * @param response     oggetto HTTP response
 * @return una mappa con il risultato del cambio password
 */
@PostMapping({ "/studente/changePassword", "/professore/changePassword" })
@ResponseBody
public Map<String, String> cambioPassword(@RequestParam String newPassword, @RequestParam String oldPassword,
                                           HttpServletRequest request, HttpServletResponse response) {
```

Figura 6.3: Esempio di uso di annotation javadoc

in questo modo per ogni funzione sono stati specificati i parametri di ingresso, ed i valori di ritorno, il tutto racchiuso all'interno dei commenti speciali necessari per poter permettere allo strumento di interpretare correttamente le etichette.

Una volta etichettate tutte le funzioni è stato necessario spostarsi nella root del progetto (in cui è situato il pom.xml) e lanciare il comando utile per generare il file HTML:

```
mvn javadoc:javadoc
```

Figura 6.4: Comando di avvio javadoc



Il risultato dell'operazione è un index.html che contiene tutte le informazioni delle interfacce documentate facilmente navigabili

```
@PostMapping("/studente/start")
public Map<String,String> startRoom(@RequestBody
                                   RoomController.startRoomBody startRoom)
                                   throws InterruptedException,
                                   ExecutionException
```

Gestione della richiesta di avvio della room.

Parameters:

startRoom - corpo della richiesta con nome classe, nome laboratorio e utente

Returns:

mappa contenente messaggio, comando e tipo di esito (success/error)

Throws:

`InterruptedException` - in caso di interruzione dell'esecuzione asincrona

`ExecutionException` - in caso di errore durante l'esecuzione asincrona

Figura 6.5: Esempio di documentazione javadoc

mentre per una consultazione più approfondita si rimanda al file situato al seguente percorso `/docs/site/apidocs/index.html`.



7. TESTING

Il testing ha l'obiettivo di individuare i difetti del sistema e verificare se il sistema soddisfa i requisiti prestabiliti. Questa pratica è servita al fine di garantire la qualità ed efficienza dell'applicazione.

Tale processo è servito al fine di individuare errori o difetti di programmazione all'interno del sistema, per un corretto impiego sono state definite una serie di strategie per scovare gli aspetti critici dell'applicazione.

7.1 Test suite per le Rest API

Nel nostro caso di studio si è scelti di usare dei test automatizzati al fine di verificare la correttezza delle API, questo approccio ci ha permesso di testare i loro parametri di input, le risposte e gli scenari d'errore.

Di seguito si riportano i piani di test di 2 API scelte:

Piano di test per la funzionalità crea room: creaRoom:

ID	DESCRIZIONE	PRECONDIZIONI	INPUT	OUTPUT ATTESI	POSTCONDIZIONI ATTESE	OUTPUT OTTENUTI	POSTCONDIZIONI OTTENUTE	ESITO
1	Classe inserita senza nome	-	classId:"", room="testroom123", descrizione="descrizione", flag="flag", yamlFile=docker-compose.yml	400:Bad request	La room non viene creata	400:Bad request	La room non è stata creata	PASS
2	Classe inserita esistente	Una classe con nome="SAD" è presente nel database	classId:"SAD", room="testroom123", descrizione="descrizione", flag="flag", yamlFile=docker-compose.yml	200:OK	La room viene creata	200:OK	La room è stata creata	PASS
3	Room inserita senza nome	-	classId:"SAD", room="", descrizione="descrizione", flag="flag", yamlFile=docker-compose.yml	400:Bad request	La room non viene creata	200:OK	La room è stata creata	NOT PASS
4	Room già esistente	Una room con nome="Testing" è presente nel database	classId:"SAD", room="Testing", descrizione="descrizione", flag="flag", yamlFile=docker-compose.yml	200:OK	La room non viene creata	200:OK	La room non è stata creata	PASS
5	Flag non inserita	-	classId:"SAD", room="Testing", descrizione="descrizione", flag="", yamlFile=docker-compose.yml	400:Bad request	La room non viene creata	400:Bad request	La room non è stata creata	PASS
6	Campo yamlFile vuoto	-	classId:"SAD", room="Testing", descrizione="descrizione", flag="flag" yamlFile= "docker-compose.yml	200:OK	La room non viene creata	200:OK	La room non viene creata	PASS
7	Formato yamlFile errato	-	classId:"SAD", room="Testing", descrizione="descrizione", flag="flag" yamlFile=testing.txt	400:Bad request	La room non viene creata	400:Bad request	La room non viene creata	PASS



Piano di test per la funzionalità start room: startRoom:

ID	DESCRIZIONE	PRECONDIZIONI	INPUT	OUTPUT ATTESI	POSTCONDIZIONI	OUTPUT OTTENUTI	POSTCONDIZIONI	ESITO
1	Start di una room con studente iscritto alla classe	La classe "SAD" esiste, la room "SpringMVC" esiste, lo studente "bob" è iscritto alla classe "SAD"	<code>{"nomeClass": "SAD", "nomeLab": "SpringMVC", "utente": "bob"}</code>	200:OK	Lo studente riceve un messaggio di conferma ed il comando per il collegamento	<code>{"msg": "Room SpringMVC avviata con successo per l'utente bob! Per il collegamento usa il comando nella box.", "type": "success", "command": "ssh -p 6367 root@localhost"}, 200</code>	La room è attiva e contattabile	PASS
2	Start di una room con studente non iscritto alla classe	La room "SpringMVC" esiste, la classe "SAD" esiste, lo studente "alice" non è iscritto alla classe "SAD"	<code>{"nomeClass": "SAD", "nomeLab": "SpringMVC", "utente": "alice"}</code>	400:Bad Request	La room non viene avviata	<code>{"msg": "Room SpringMVC avviata con successo per l'utente alice! Per il collegamento usa il comando nella box.", "type": "success", "command": "ssh -p 25188 root@localhost"}, 200</code>	La room è attiva e contattabile	NOT PASS
3	Start di una room senza effettuare prima lo stop di una room già avviata dallo stesso utente	La room "SpringMVC" esiste, la classe "SAD" esiste, lo studente "alice" non è iscritto alla classe "SAD"	<code>{"nomeClass": "SAD", "nomeLab": "SpringMVC", "utente": "bob"}</code>	200:OK	La room viene avviata con successo rimuovendo la precedente	<code>{"msg": "Room SpringMVC avviata con successo per l'utente bob! Per il collegamento usa il comando nella box.", "type": "success", "command": "ssh -p 6367 root@localhost"}, 200</code>	La room è attiva e contattabile	PASS
4	Start di una room da un utente inesistente	Lo studente "belzebu" non esiste	<code>{"nomeClass": "SAD", "nomeLab": "SpringMVC", "utente": "belzebu"}</code>	400:Bad Request	La room non viene avviata	<code>{"msg": "Room SpringMVC avviata con successo per l'utente belzebu! Per il collegamento usa il comando nella box.", "type": "success", "command": "ssh -p 35328 root@localhost"}, 200</code>	La room è attiva e contattabile	NOT PASS
5	Start di una room inesistente	La room "DirittoRomano" non esiste	<code>{"nomeClass": "SAD", "nomeLab": "DirittoRomano", "utente": "bob"}</code>	200:OK	La room non viene avviata	<code>{"msg": "Errore durante l'avvio del laboratorio, riprovare più tardi", "type": "error"}, 200</code>	La room non viene avviata	PASS

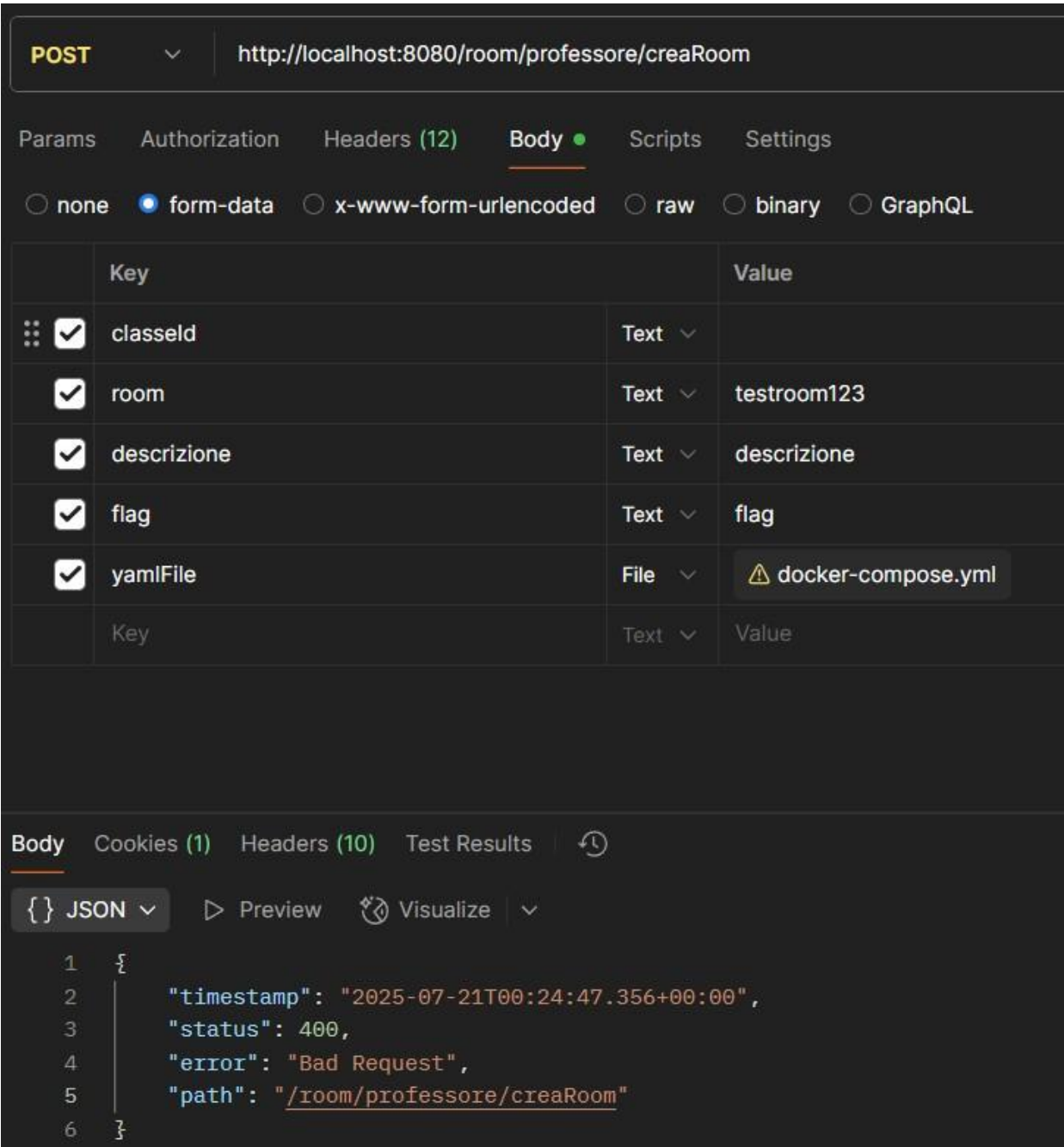


7.2 Testing con Postman

Il testing concreto delle API è stato sviluppato attraverso Post-man, ossia un'applicazione che consente di individuare richieste http e visualizzare le risposte in modo interattivo. In particolare, consente di verificare se le risposte sono conformi alle aspettative, semplificando il processo di controllo di qualità dell'API.

7.2.1 Testing creaRoom

TC1 classe senza nome:



The screenshot shows the Postman interface for a POST request to `http://localhost:8080/room/professore/creaRoom`. The request body is set to `form-data` and contains five fields: `classeld`, `room`, `descrizione`, `flag`, and `yamlFile`. The `yamlFile` field is set to `docker-compose.yml`. The response is a `400 Bad Request` with the following JSON body:

```
{
  "timestamp": "2025-07-21T00:24:47.356+00:00",
  "status": 400,
  "error": "Bad Request",
  "path": "/room/professore/creaRoom"
}
```




TC2: classe valida

POST ▼ http://localhost:8080/room/professore/creaRoom

Params Authorization Headers (12) **Body ●** Scripts Settings

☐ none ☒ form-data ☐ x-www-form-urlencoded ☐ raw ☐ binary ☐ GraphQL

<input checked="" type="checkbox"/>	classeld	Text ▼	test
<input checked="" type="checkbox"/>	room	Text ▼	testroom
<input checked="" type="checkbox"/>	flag	Text ▼	test
<input checked="" type="checkbox"/>	descrizione	Text ▼	testdesc
<input checked="" type="checkbox"/>	yamlFile	File ▼	docker-compose.yml
	Key	Text ▼	Value

Body Cookies (1) Headers (11) Test Results

{ } JSON ▼ ▶ Preview Visualize ▼

```
1  {
2    "message": "Laboratorio creato con successo.",
3    "type": "success"
4  }
```



TC3: room esistente

POST ▼ http://localhost:8080/room/professore/creaRoom

Params Authorization Headers (12) **Body** ● Scripts Settings

☐ none ☒ form-data ☐ x-www-form-urlencoded ☐ raw ☐ binary ☐ GraphQL

<input checked="" type="checkbox"/>	classId	Text ▼	test
<input checked="" type="checkbox"/>	room	Text ▼	testRoom
<input checked="" type="checkbox"/>	descrizzone	Text ▼	test
<input checked="" type="checkbox"/>	flag	Text ▼	test
<input checked="" type="checkbox"/>	yamlFile	File ▼	docker-compose.yml
	Key	Text ▼	Value

Body Cookies (1) Headers (10) Test Results ↺

{ } JSON ▼ ▶ Preview 🔗 Visualize ▼

```
1 {
2   "timestamp": "2025-07-21T00:31:09.466+00:00",
3   "status": 400,
4   "error": "Bad Request",
5   "path": "/room/professore/creaRoom"
6 }
```



TC4: room non inserita

HTTP New Collection / Room senza nome

POST ▼ `http://localhost:8080/room/professore/creaRoom`

Params Authorization Headers (12) **Body** ● Scripts Settings

☐ none ☒ form-data ☐ x-www-form-urlencoded ☐ raw ☐ binary ☐ GraphQL

<input checked="" type="checkbox"/>	classeld	Text ▼	test
<input checked="" type="checkbox"/>	room	Text ▼	
<input checked="" type="checkbox"/>	descrizione	Text ▼	testdesc
<input checked="" type="checkbox"/>	flag	Text ▼	test
<input checked="" type="checkbox"/>	yamlFile	File ▼	docker-compose.yml
	Key	Text ▼	Value

Body Cookies (1) Headers (11) Test Results ↺

{} JSON ▼ ▶ Preview 🔄 Visualize ▼

```
1 {  
2   "message": "Laboratorio creato con successo.",  
3   "type": "success"  
4 }
```



TC5: campo flag vuoto

POST ▼ http://localhost:8080/room/professore/creaRoom

Params Authorization Headers (12) **Body** ● Scripts Settings

☐ none ☒ form-data ☐ x-www-form-urlencoded ☐ raw ☐ binary ☐ GraphQL

<input checked="" type="checkbox"/>	classeld	Text ▼	test
<input checked="" type="checkbox"/>	room	Text ▼	test1
<input checked="" type="checkbox"/>	descrizione	Text ▼	testdesc
<input checked="" type="checkbox"/>	flag	Text ▼	
<input checked="" type="checkbox"/>	yamlFile	File ▼	⚠ docker-compose.yml
	Key	Text ▼	Value

Body Cookies (1) Headers (10) Test Results ↺

{} **JSON** ▼ ▶ Preview 🔍 Visualize ▼

```
1 {  
2   "timestamp": "2025-07-21T00:32:09.166+00:00",  
3   "status": 400,  
4   "error": "Bad Request",  
5   "path": "/room/professore/creaRoom"  
6 }
```




TC6: formato yamle file errato

POST ▼ <http://localhost:8080/room/professore/creaRoom>

Params Authorization Headers (12) **Body** ● Scripts Settings

☐ none ☒ form-data ☐ x-www-form-urlencoded ☐ raw ☐ binary ☐ GraphQL

<input checked="" type="checkbox"/>	classeld	Text ▼	test
<input checked="" type="checkbox"/>	room	Text ▼	test1
<input checked="" type="checkbox"/>	descrizione	Text ▼	testdesc
<input checked="" type="checkbox"/>	flag	Text ▼	testflag
<input checked="" type="checkbox"/>	yamiFile	File ▼	 structure.txt
	Key	Text ▼	Value

Body Cookies (1) Headers (11) Test Results ↺

{ } JSON ▼ ▶ Preview 🔍 Visualize ▼

```
1 {
2   "message": "File non valido. Deve essere un file .yaml o .yml",
3   "type": "error"
4 }
```



TC7: formato yamleFile corretto

POST ▼ http://localhost:8080/room/professore/creaRoom

Params Authorization Headers (12) **Body ●** Scripts Settings

☐ none ☒ form-data ☐ x-www-form-urlencoded ☐ raw ☐ binary ☐ GraphQL

<input checked="" type="checkbox"/>	classeld	Text ▼	test
<input checked="" type="checkbox"/>	room	Text ▼	testroom
<input checked="" type="checkbox"/>	flag	Text ▼	test
<input checked="" type="checkbox"/>	descrizione	Text ▼	testdesc
<input checked="" type="checkbox"/>	yamleFile	File ▼	docker-compose.yml
	Key	Text ▼	Value

Body Cookies (1) Headers (11) Test Results

{ } JSON ▼ ▶ Preview Visualize ▼

```
1 {
2   "message": "Laboratorio creato con successo.",
3   "type": "success"
4 }
```



7.2.2 Testing creaRoom

TG1: studente iscritto alla classe

The screenshot shows the Postman interface with a workspace named 'AvvioLaboratorioStudenteIscritto - My Workspace'. A collection named '/room/studente/start' is expanded, showing a POST request to 'http://localhost:8080/room/studente/start'. The request body is a JSON object:

```
{ "nomeClass": "SAD", "nomeLab": "SpringMVC", "utente": "bob" }
```

. The response is a 200 OK status with a response time of 13.62 s and a body of:

```
{ "msg": "Room SpringMVC avviata con successo per l'utente bob! Per il collegamento usa il comando nella box.", "type": "success", "command": "ssh -p 6367 root@localhost" }
```

TG2: studente non iscritto alla classe

The screenshot shows the Postman interface with a workspace named 'AvvioLaboratorioStudenteNonIscritto - My Workspace'. A collection named '/room/studente/start' is expanded, showing a POST request to 'http://localhost:8080/room/studente/start'. The request body is a JSON object:

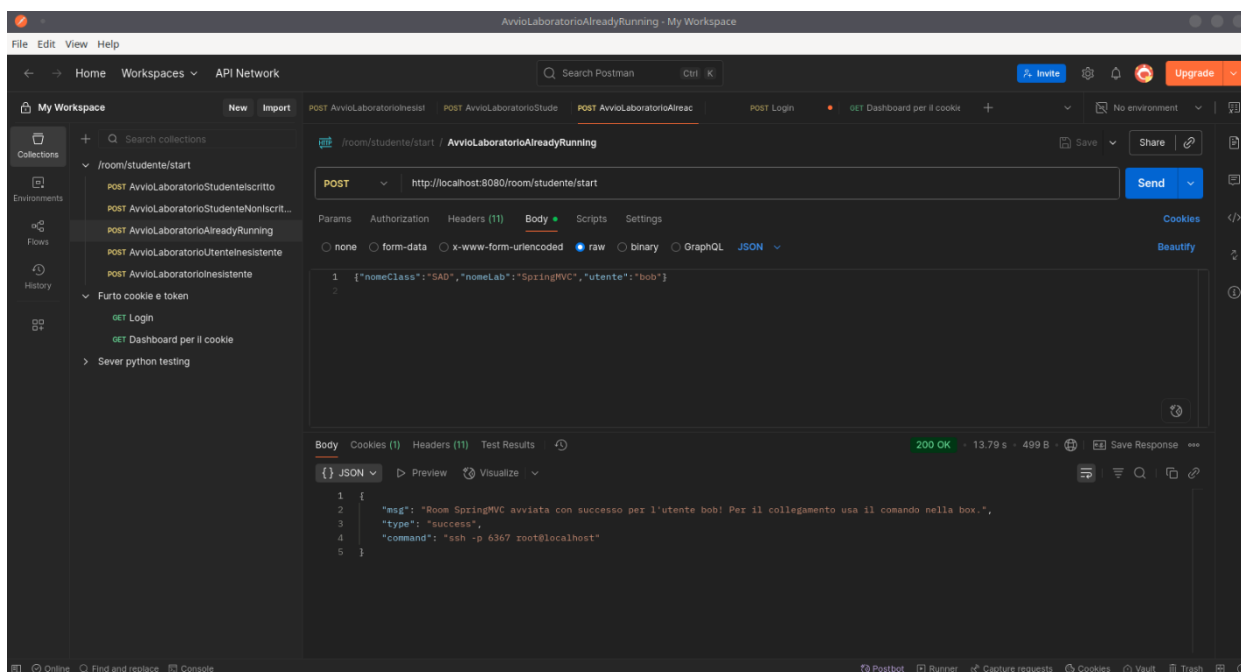
```
{ "nomeClass": "SAD", "nomeLab": "SpringMVC", "utente": "alice" }
```

. The response is a 200 OK status with a response time of 12.44 s and a body of:

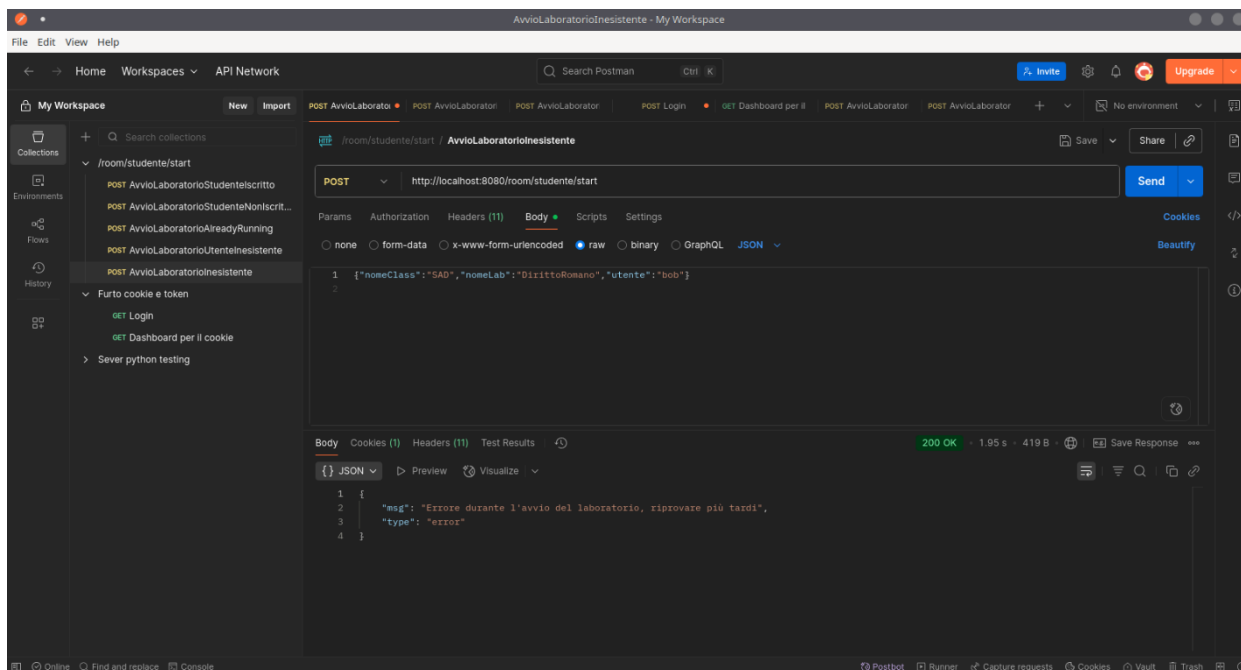
```
{ "msg": "Room SpringMVC avviata con successo per l'utente alice! Per il collegamento usa il comando nella box.", "type": "success", "command": "ssh -p 25188 root@localhost" }
```



TG3: start room prima di stoppare una in esecuzione



TG4: start room da parte di un utente inesistente





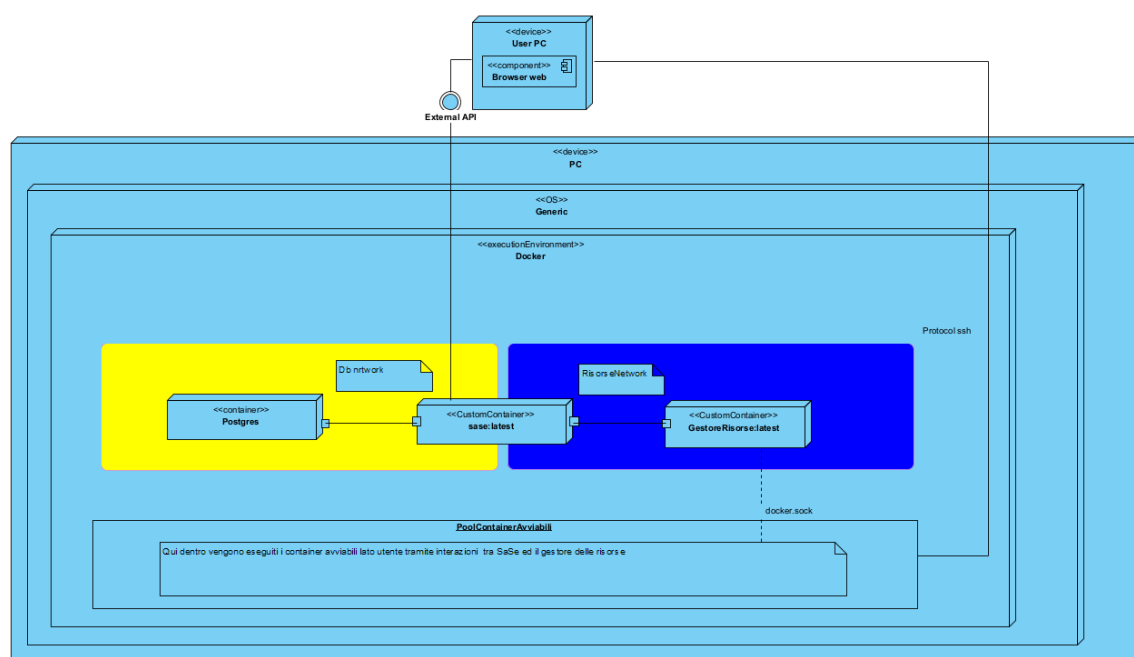
8. CONTAINERIZZAZIONE

Al fine di semplificare l'esecuzione dell'applicazione si è scelto di containerizzarla permettendone l'avvio completo tramite un singolo comando.

In questo caso, il *Gestore delle risorse* ha la necessità di gestire a sua volta altri container Docker. Di conseguenza, una semplice containerizzazione non sarebbe stata sufficiente, poiché non avrebbe garantito il corretto funzionamento di tale componente.

La soluzione adottata è l'utilizzo dell'approccio Docker-in-Docker (DinD), che consente al container principale di accedere al *Docker daemon*. In questo modo, il container può comunicare con l'esterno come un normale container, ma allo stesso tempo ha la possibilità di avviare e gestire altri container internamente.

Di seguito viene presentato il diagramma di deploy dell'applicazione dockerizzata:



In questo modo è possibile **simulare un ambiente reale** utilizzando più container, ciascuno dei quali rappresenta un potenziale server o nodo su cui distribuire le diverse componenti dell'applicazione.

Questo approccio consente di modellare fedelmente l'architettura distribuita del sistema, così come verrà eseguita in produzione.



9. SVILUPPI FUTURI

Di seguito vengono proposti una serie di punti interessanti per eventuali implementazioni future:

- **Implementazione di ulteriori meccanismi di gamification**
L'introduzione di meccanismi di gamification, come badge, riconoscimenti, e classifiche settimanali, ha lo scopo di aumentare l'engagement e la partecipazione attiva degli utenti. Premiare comportamenti virtuosi o risultati ottenuti stimola una competizione sana e migliora l'esperienza d'uso della piattaforma.
- **Espansione del dominio dei laboratori**
La soluzione proposta prevede un focus specifico sulla security che rappresenta la mission della piattaforma. Essendo il contesto dell'applicazione un target e non un vincolo al funzionamento del sistema, si potrebbe pensare di aumentare il range di laboratori inserendo nuove sezioni dedicate ad altri argomenti.
- **Servizi di sicurezza e pentesting**
L'applicazione non è pensata per essere robusta per cui potrebbe essere utile effettuare dei controlli di sicurezza con strumenti e routine per eseguire test di vulnerabilità automatici (come XSS, CSRF, SQLi), per garantire la sicurezza del sistema. Questo sviluppo mira a offrire un ambiente controllato per la simulazione di attacchi e la valutazione della resilienza delle applicazioni ospitate.
- **Implementazione di un server proxy con IP pubblico per il load balancing**
Per gestire un numero crescente di richieste e sessioni simultanee, si potrebbe pensare di introdurre un reverse proxy con indirizzo IP pubblico che svolga anche funzioni di load balancing tra più nodi o container. Questo garantirà una migliore scalabilità e distribuzione del carico.
- **Routine di sanificazione automatica delle room**
Per evitare il sovraccarico del sistema, sarebbe bene implementare delle routine automatiche di pulizia che chiudano le room rimaste aperte troppo a lungo o inattive. Questa misura migliorerà la gestione delle risorse e limiterà i rischi legati a sessioni lasciate incustodite.
- **Miglioramento della robustezza delle operazioni sul database**
Un altro sviluppo potenziale è l'introduzione di meccanismi di gestione delle eccezioni e rollback più sofisticati per le operazioni critiche sul database. Questo assicurerà una maggiore affidabilità e integrità dei dati, soprattutto in scenari concorrenti o in presenza di errori di rete.
- **Aggiornamento della funzione di calcolo dello score**
La funzione di score è pensata per calcolare uno score su 120 minuti e non c'è possibilità di modificare questo limite se non direttamente nel codice. Un'idea potrebbe essere quella di implementare una funzionalità che permetta al docente di scegliere questi parametri o di inserire nuove tipologie di calcolo.



A. Guida agli sviluppatori

Si riporta questa sezione nel caso in cui si volesse modificare il codice dell'applicazione in modo da fornire indicazioni utili a seguire una routine di coding che permetta di modificare i container senza problemi.

In particolare si fa notare che:

1. La repository del progetto presenta un *Dockerfile* che rappresenta il file che permette di buildare il container dell'applicazione spring. Rappresenta il punto di partenza per effettuare le modifiche al codice Spring MVC del webserver sase.
2. I container utilizzati per il gestore delle risorse e per le room di test sono customizzate e presenti su dockerhub all'indirizzo <https://hub.docker.com/repositories/veturo98>.



B. Guida alla scrittura dei laboratori per i Professori

Una parte fondamentale di questo progetto è la presenza di laboratori accessibili dall'esterno e di esperti pronti ad offrire le proprie competenze nella creazione di container con le challenge più fantasiose possibili.

Il progetto presenta nel documento SAD-exam.yaml un esempio di challenge che può essere utile a comprendere quale sia il modo corretto di scrivere un laboratorio.

In questa sezione verranno presentate le caratteristiche dei docker-compose.yml validi ad essere dei laboratori di SAD-Security.

Di seguito viene mostrato il laboratorio di esempio:

```
! SAD-exam.yaml
1  services:
2
3      contact-me:
4          image: veturo98/server-ssh:esame
5          restart: always
6          command: /usr/sbin/sshd -D # Start server ssh
7          working_dir: /home/ubuntu/
8          healthcheck:
9              test: ["CMD", "echo", "test"]
10             interval: 10s
11             timeout: 5s
12             retries: 2
13         ports:
14             - ${PORTA}:22
15
16
17         spring-mvc:
18             image: veturo98/server-ssh:flag
19             restart: always
20             command: /usr/sbin/sshd -D # Start server ssh
21             working_dir: /home/ubuntu/
22             healthcheck:
23                 test: ["CMD", "echo", "test"]
24                 interval: 10s
25                 timeout: 5s
26                 retries: 2
```

Figura A.1



Si fa notare che:

- Deve essere sempre presente almeno un laboratorio che esponga una porta all'esterno del tipo `${PORTA}`; questo parametro viene gestito dal gestore delle risorse internamente e deve essere lasciato esattamente in questo modo.
- Per essere contattato in ssh dall'esterno, il container che espone la porta deve avviare un server ssh sulla porta 22; in caso contrario verrà avviato un laboratorio non contattabile da nessuno!
- La rete sulla quale viene avviato il container è virtuale per cui non c'è alcun rischio di overlap a patto di non fissare gli IP dei laboratori; un errore di questo tipo può generare degli internal error sul server gestore delle risorse.
- Essendo un docker compose file si può pensare di caricare laboratori che riguardano il networking e di far collegare l'utente alla "rete locale" attraverso il server ssh; in alternativa si può pensare di costruire laboratori preparando i container e facendo eseguire i comandi solamente sul container *ponte*.