POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Informatica

Tesi di Laurea Magistrale

Gamification per lo sviluppo e testing di codice Java

Relatori

Prof. Marco TORCHIANO

Prof. Riccardo COPPOLA

Dott. Tommaso FULCINI

Candidato

Antonio MATERAZZO

Aprile 2023

Sommario

L'attività del *testing* ha assunto un ruolo fondamentale per verificare la qualità e la correttezza del software, risultando, però, spesso trascurata in ambito didattico, con studenti che vi si approcciano controvoglia a causa della sua ripetitività.

Questo lavoro di tesi mira a sfruttare la *gamification* per aumentare il coinvolgimento degli studenti nei confronti di tale attività, presentando loro dinamiche come competizione, espressione di sé e miglioramento personale.

È stata quindi sviluppata Unit Brawl, un'applicazione con integrati elementi di gamification pensata per supportare l'insegnamento dello unit testing nel corso di "Programmazione a oggetti", appartenente al corso di laurea di 1° livello in Ingegneria Informatica presso il Politecnico di Torino. L'applicazione è pensata per gestire più round, ciascuno dei quali composto da studenti che sviluppano programmi Java e unit test da eseguire gli uni sugli altri. I giocatori guadagnano punti scrivendo codice corretto che non faccia fallire i test degli altri giocatori, o scrivendo test in grado di rilevare difetti nel codice degli avversari.

I risultati di una validazione preliminare per valutare funzionalità e prestazioni di Unit Brawl sembrano promettenti, comprovando la stabilità dell'implementazione attuale. Di conseguenza, è stata pianificata una validazione con gli studenti, finalizzata a verificare l'efficacia degli elementi di gioco nell'incrementare l'interesse e migliorare il loro apprendimento circa l'argomento del testing.

Ringraziamenti

Con questa tesi si conclude – almeno momentaneamente – un lungo percorso di studi, di cui ogni tassello ha contribuito a formare parte della persona che sono ora. Per questo motivo ringrazio tutti coloro che hanno preso parte a questo viaggio, chi più frequentemente, chi solo di sfuggita.

In particolare, ci tengo a ringraziare i professori Marco Torchiano e Riccardo Coppola e il dott. Tommaso Fulcini, che mi hanno concesso l'opportunità di approfondire un argomento con cui ho sempre desiderato interfacciarmi e sul quale spero di poter tornare in futuro. Vi ringrazio per i consigli e le indicazioni fornitemi durante la scrittura della tesi e in particolar modo per avermi dato la possibilità di concludere questo percorso con la stesura di "Survival of the Tested: Gamified Unit Testing Inspired by Battle Royale", articolo accademico basato sul lavoro di questa tesi.

Le persone che voglio ringraziare principalmente sono papà e mamma: non si può esprimere a parole la gratitudine per tutti i sacrifici fatti nel corso di questi anni e il supporto fornitomi in tutte le mie scelte, ovunque mi trovassi.

Gianluca andrebbe ringraziato per infinite ragioni: qui voglio ringraziarti per avermi fatto sentire (e per farmi sentire tuttora) vicino a te in ogni momento, indipendentemente da dove ci troviamo. Avrò sempre bisogno di te.

Penso sia normale guardare alle scelte prese nel corso della propria vita con titubanza, perché ogni scelta che compiamo ci impedisce di conoscere come sarebbero state le alternative. Nonostante questo, sono sicuro di non avere alcun ripensamento sulla scelta di venire a Torino, perché qui mi sono avvicinato a Claudia. Grazie per i bei momenti che abbiamo condiviso insieme e per tutti quelli che ci attendono, grazie soprattutto per essere stata con me nei momenti meno facili. "Andiamo".

Ringrazio i nonni, parenti e amici che mi sono stati vicino in questi anni: Fabiola e Santo per le lezioni, gite e giornate passate insieme a Torino; Fabiola e Claudia, siete le migliori coinquiline che un Gerardo possa desiderare; Paola e Miriam per tutte le organizzatissime organizzazioni e risate tra Lamezia, Cartolano, Parma, Pavia e Torino; Giovanni per le "chiamatine" internazionali e ristoratrici; Ferdinando già lo sai, non c'è bisogno di aggiungere altro; Antonio, Nicola e Shizuka – Sensei, Kenzo e Jake8 – per aver deciso di condividere insieme un progetto che ci unisce e

che da anni vi rende parte della mia quotidianità.

Per concludere, voglio ringraziare mia cugina Roberta. Forse ancora non te ne rendi conto, ma ogni volta che siamo insieme ci rendi persone migliori.

A me stesso auguro di vivere ogni momento, persona, traguardo o caduta che incontrerò negli anni a venire.

Indice

Elenco delle tabelle

Elenco delle figure

Acronimi

BPMN
: Business Process Model and Notation

CI
: Continuous Integration

PO
: Programmazione a Oggetti

SUT
: System Under Test

UML
: Unified Modeling Language

Capitolo 1

Introduzione

Al giorno d'oggi sono sempre di più gli ambiti della nostra quotidianità ad essere supportati dall'utilizzo diretto o indiretto di qualche tipo di software: preso atto della situazione, si può dedurre come un maggiore e migliore supporto sia strettamente legato alla qualità e alla correttezza del software stesso. Tali aspetti possono essere esaminati per mezzo del testing, che consiste nella verifica del corretto comportamento di un sistema o di un suo componente in modo da scovare eventuali discordanze con i criteri prestabiliti.

Nonostante queste premesse, l'attività di testing è spesso trascurata, rimandata o eseguita in maniera approssimativa [6], comportando, tra le altre cose, insoddisfazione degli utenti, interruzione di servizi, perdite economiche e nei casi peggiori rischi alla pubblica sicurezza. Una ricerca di Tricentis [2] risalente al 2016 afferma che negli Stati Uniti D'America i difetti nei software costano circa mille miliardi in risorse. Tali numeri non stupiscono se si considerano le mancanze legate al testing come parte di quel più vasto aspetto che è il *technical debt*: ogni imperfezione nel software o nel suo processo di sviluppo potrebbe provocare, se non adeguatamente gestita, un accumulo di complessità che alla lunga comporterebbe gravi rallentamenti al processo di sviluppo e conseguente perdita di denaro. Segue qualche esempio di gravi conseguenze causate dalla presenza di difetti non rilevati in alcuni software:

- un bug nel sistema di controllo della radioterapia Therac-25 comportò la morte di almeno cinque pazienti in seguito alla somministrazione di un eccessivo quantitativo di radiazioni;

- nel 1996 il razzo Ariane 5 esplose a circa 40 secondi dalla sua partenza a causa di un bug (un errore di conversione di dati a 64 e 16 bit) non intercettato dai test poiché non effettuati a dovere;

- nel 1998 la navicella spaziale Mars Climate Orbiter si incendiò dopo essersi avvicinata eccessivamente alla superficie marziana a causa di un errore di conversione di unità di misura, causando alla NASA una perdita di 125 milioni di dollari;

- nel 1991 un bug impedì al sistema anti-balistico di una base americana in Arabia Saudita di intercettare in tempo un missile iracheno: un ritardo di 0.33 secondi in un calcolo causò la morte di 28 soldati americani.

In ambito accademico, l'insegnamento dei concetti legati al testing risulta spesso assente o messo in secondo piano rispetto agli altri relativi alle basi dell'informatica; gli stessi studenti, inoltre, si sentono poco motivati a ideare ed eseguire test a causa della natura spesso ripetitiva e poco creativa del compito [23] [12]. Tuttavia, la scrittura di codice di scarsa qualità comporta la necessità di impiegare risorse (economiche e temporali) per scovare e correggere difetti che potrebbero passare inosservati anche per lungo tempo. Si deduce quindi come sia fondamentale l'introduzione di metodologie efficaci per l'insegnamento del testing all'interno di corsi accademici appositi [17].

Lo scopo di questo lavoro di tesi è quello di migliorare la qualità di apprendimento degli studenti in merito a concetti legati all'attività del testing, altrimenti trascurata o portata avanti a stento, rendendola al contrario appagante e interessante. Per fare ciò verranno sfruttati elementi di gamification, definita come l'applicazione di concetti, principi e meccaniche di game design in ambiti non ludici. Il fine principale della gamification è quello di incrementare il coinvolgimento degli utenti nello svolgere determinate attività che altrimenti avrebbero difficilmente portato avanti o influenzare il loro comportamento.

La gamification è ormai presente in moltissimi ambiti della nostra quotidianità, da quello aziendale a quello medico, passando in particolar modo per servizi e marketing, ottenendo benefici non indifferenti nei casi in cui le meccaniche vengano applicate e integrate fra loro in maniera ragionevole e corretta. In letteratura non sono pochi i tool che sfruttano la gamification per l'insegnamento di argomenti relativi in generale all'informatica, ma anche al testing nello specifico. Molti di essi riportano risultati soddisfacenti in seguito agli esperimenti effettuati [13], nonostante esistano alcuni studi che evidenziano conseguenze negative o indesiderate [3].

Oggetto di questo lavoro di tesi è la progettazione, sviluppo e analisi di **Unit Brawl**, un'applicazione multigiocatore per lo sviluppo di codice Java e dei rispettivi unit test black-box e white-box, con al suo interno elementi di gamification. L'applicazione è destinata a supportare i laboratori svolti dagli studenti durante tutta la durata del corso di "Programmazione a oggetti" (PO), tenuto durante il corso di laurea di 1° livello in Ingegneria Informatica presso il Politecnico di Torino. È importante sottolineare che la sua integrazione avverrà in ambiente educativo: la presenza di un contesto così delicato sarà alla base di alcune scelte progettuali

riguardanti importanti meccaniche di gamification. Si precisa inoltre che Unit Brawl può essere implementata anche da altre realtà riguardanti l'insegnamento di concetti relativi all'ingegneria del software slegate dal contesto del corso di PO, purché vengano rispettati vincoli e regole spiegati nelle prossime sezioni.

In particolare, la tesi seguirà la seguente struttura:

- nel Capitolo 2 ("Background") verranno approfonditi gli ambiti coinvolti dalla tesi, ovvero quelli di testing del software (descrizione, fasi del processo e tipologie) e gamification (con particolare attenzione alla descrizione del framework di riferimento scelto);
- nel Capitolo 3 ("Design") verranno illustrati i processi e le regole che governano la piattaforma e in seguito elencate e motivate le meccaniche di gamification adottate;
- nel Capitolo 4 ("Implementazione") si tratterà l'implementazione dell'applicazione in ogni suo dettaglio, illustrandola per mezzo del linguaggio tipico dell'ingegneria del software;
- nel Capitolo 5 ("Validazione") verrà descritto l'esperimento di validazione della piattaforma e discussi i risultati ottenuti;
- si concluderà con il Capitolo 6 ("Conclusioni"), in cui si rifletterà sullo stato attuale di Unit Brawl e sui suoi limiti, discutendo infine dei possibili sviluppi futuri.

Capitolo 2

Background

In questo capitolo verranno approfonditi gli ambiti coinvolti nello sviluppo di Unit Brawl: il testing del software (che verrà introdotto e definito nelle sue fasi e tipologie) e la gamification (con particolare attenzione al framework Octalysis scelto come riferimento in fase di design).

2.1 Software Testing

Il concetto di testing del software può essere introdotto mediante le seguenti definizioni riportate dal "IEEE Standard Glossary of Software Engineering Terminology" [14]:

- **test**, "attività in cui un sistema o un suo componente è eseguito secondo specifiche condizioni, in cui i risultati vengono osservati e registrati e in cui viene svolta una valutazione di alcuni aspetti del sistema o componente";
- **verifica**, "il processo di valutazione di un sistema o di un suo componente per determinare se il prodotto di una determinata fase di sviluppo soddisfi le condizioni imposte all'inizio di quella fase";
- **validazione**, "il processo di valutazione di un sistema o di un suo componente durante o alla fine del processo di sviluppo per determinare se rispetta determinati requisiti".

Tenendo a mente queste tre definizioni, è chiaro come lo scopo principale del testing del software sia quello di verificare il corretto comportamento di un'applicazione o di un suo componente per mezzo dell'esecuzione, osservazione, registrazione e valutazione dei risultati ottenuti. L'utilità di tale attività è indiscutibile poiché permette di rintracciare la presenza di difetti (di design o di sviluppo) all'interno dell'applicazione, permettendo quindi di risolverli e minimizzando l'impatto che avrebbe causato il rilascio di un prodotto mal funzionante.

2.1.1 Fasi del processo

Le fasi del processo di testing, generalmente facenti parte del **"Software Testing Life Cycle"**, sono: analisi dei requisiti, pianificazione dei test, sviluppo dei test case, configurazione dell'ambiente di esecuzione, esecuzione dei test, chiusura. Nella prima fase vengono raccolti i requisiti che possono essere sottoposti a test, da un punto di vista interno e per mezzo del dialogo con gli stakeholders. Si passa poi – nella fase di pianificazione – a definire una stima dei costi derivanti dalla fase di test, unitamente alla strategia da adottare per lo sviluppo degli stessi. Tale strategia sarà fondamentale per lo sviluppo dei test case, della loro verifica e della preparazione dei dati da sottoporre ai test. La fase di esecuzione dei test e di report dei bug viene anticipata da quella di configurazione dell'ambiente di esecuzione, durante la quale si decidono i dettagli software e hardware di esecuzione dei test. Il processo si chiude con la strutturazione e analisi dei risultati ottenuti.

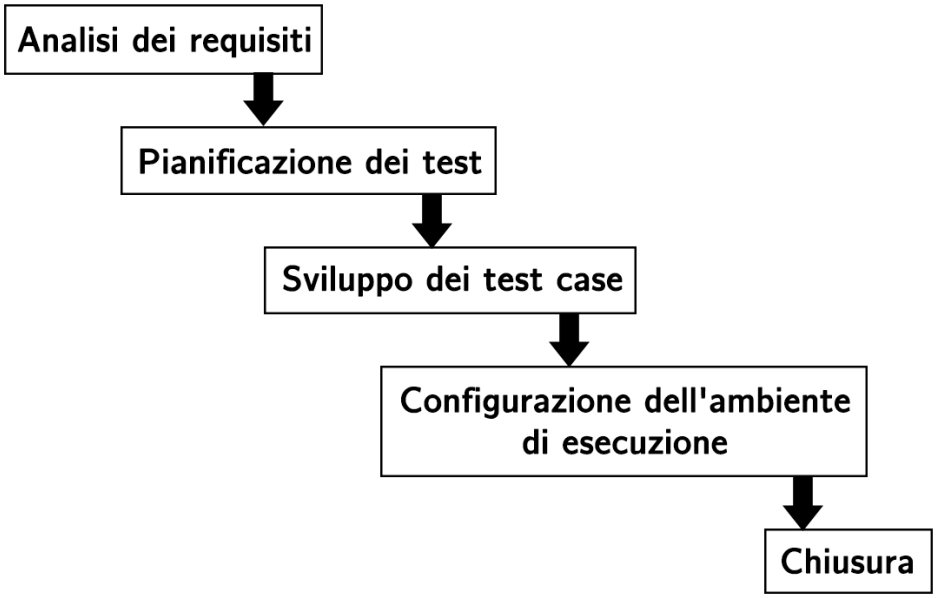

Figura 2.1: Rappresentazione del "Software Testing Life Cycle".

Un altro approccio noto è quello dello **sviluppo guidato dai test** (meglio conosciuto come "Test Driven Development" o "TDD"), per cui lo sviluppo di un programma deve seguire lo sviluppo dei test e deve essere volto al loro soddisfacimento. Secondo questa metodologia, i requisiti che caratterizzano nuove funzionalità da implementare devono essere convertiti in test che li rispecchino; a questo punto si può passare alla scrittura di codice volto unicamente a superare tali test. Nel suo libro "Test-Driven Development: By Example" [5], Kent Beck individua due principi che possono guidare un intero TDD: scrivere nuovo codice solo in seguito

al fallimento di un test e successivamente pensare al refactoring. Il primo passo è quindi quello di scrivere il test secondo i requisiti identificati e confermarne un esito fallimentare vista l'iniziale assenza di codice che lo soddisfi; si passa poi all'implementazione della funzione da testare, nella maniera più diretta possibile e fino a ottenere un esito positivo del test; solo in seguito si esegue un refactoring del codice, purché non invalidi l'esito del test. In tal modo ogni programmatore diventa responsabile di una prima verifica di correttezza del proprio operato, riducendo la presenza di errori e permettendo a tutto il processo di incontrare meno ostacoli indesiderati. Lo sviluppo guidato dai test è particolarmente sfruttato nell'Extreme Programming, metodologia agile incentrata sulla produzione di software di qualità che soddisfi i requisiti richiesti, rispondendo rapidamente a eventuali cambiamenti degli stessi.

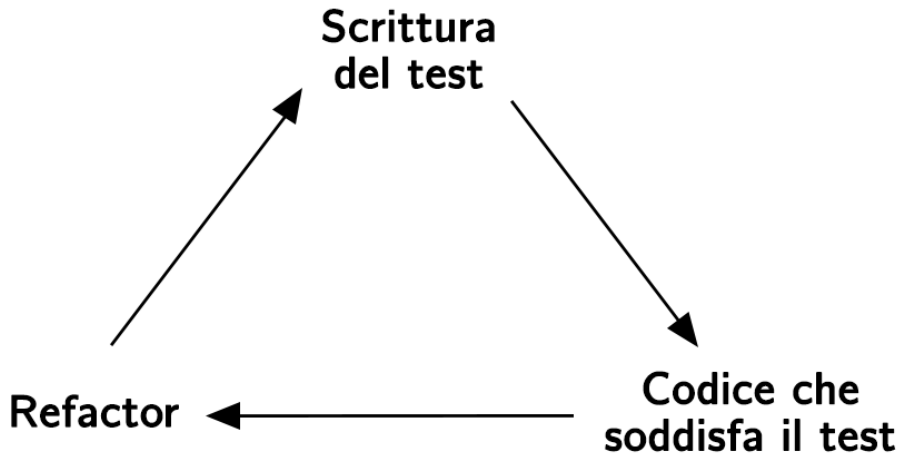

Figura 2.2: Rappresentazione dello sviluppo guidato dai test.

2.1.2 Tipologie di test

Esistono vari tipi di test con esiti e ambiti di azione differenti. Una prima categorizzazione riguarda il livello di specificità:

- **Unit testing** , eseguito per valutare il corretto funzionamento di singole unità del software;
- **Integration testing**, combina i componenti e valuta le interazioni fra essi;
- test dell'**interfaccia utente**, si concentra sull'interfaccia utente e sulle possibili interazioni fra l'utente e il sistema;

- **System testing**, eseguito sul sistema al completo;

- **Acceptance testing**, strettamente legato al rapporto con clienti o altre parti interessate, permette di dimostrare il soddisfacimento dei criteri di accettazione da parte del prodotto.

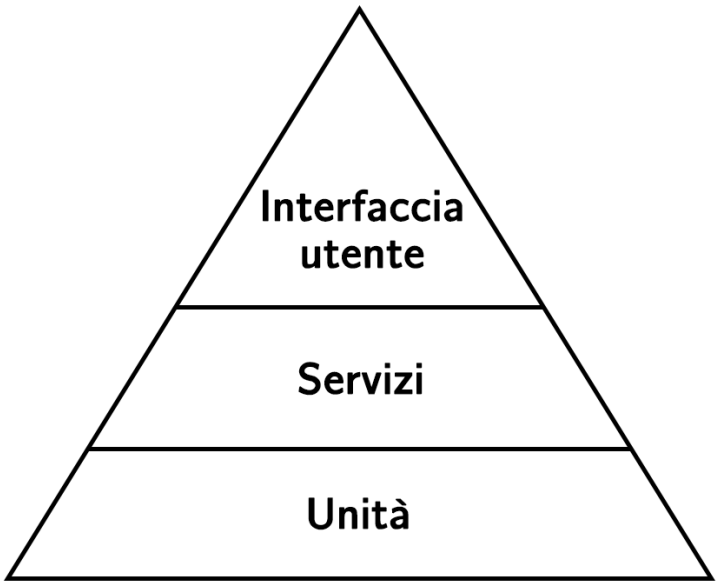

Figura 2.3: Piramide di automazione dei test.

Una strategia di esecuzione di questi tipi di test è stata definita come "piramide di automazione dei test" da Mike Cohn nel suo libro "Succeeding With Agile" [7]: si inizia dagli unit test (alla base della piramide poiché molto più numerosi degli altri) in modo da intercettare con precisione i primi bug a livello dei singoli moduli; si prosegue poi con il livello dei servizi, ovvero quello riguardante i test delle funzionalità offerte dal sistema (integration test); si conclude con il test dell'interfaccia utente, molto più complesso e dispendioso dei precedenti a causa di tutte le interazioni tra componenti che possono verificarsi. L'autore del libro sottolinea come il passaggio dal livello dei servizi permetta di snellire efficacemente i test dell'interfaccia utente, che possono così concentrarsi principalmente sulla verifica del corretto collegamento fra componenti grafici e funzionalità piuttosto che sul testing dei servizi stessi.

Un'ulteriore suddivisione si può effettuare in base alle tecniche di test impiegate. Seguono alcuni esempi:

- test **black-box** o **funzionale**, ignora le caratteristiche interne del componente o applicazione e consiste unicamente nell'analizzare i dati in uscita in funzione di quelli di ingresso scelti;
- test **white-box**, esamina il comportamento del componente o applicazione in seguito all'esecuzione di tutte le istruzioni, sentieri o diramazioni derivanti da punti di decisione all'interno del codice;
- **mutation testing**, sottoposti a più versioni dello stesso programma, hanno lo scopo di scovare eventuali mutazioni (alterazioni del programma originale);
- test **esplorativo**, permette all'utente di esplorare liberamente il sistema aumentando il suo livello di conoscenza dello stesso durante il processo;

Lo scopo del testing black-box è quello di verificare la correttezza funzionale del sistema ("System Under Test" o SUT), mentre il testing white-box è atto a verificare parametri legati all'analisi del codice a cui si ha accesso. Con il mutation testing vengono introdotti cambiamenti all'interno del codice sorgente e successivamente eseguiti dei test su di esso, in modo da verificare la qualità di questi ultimi in base alla loro capacità di scovare o meno le differenze rispetto all'originale. Il testing esplorativo è svolto da utenti in maniera incrementale, in modo da permettere loro di esplorare l'interfaccia utente a piacimento in base alla conoscenza acquisita.

Infine vengono elencati alcuni tipi di test in base allo scopo che si vuole raggiungere o all'aspetto da analizzare:

- test delle **prestazioni**, per valutare il soddisfacimento di requisiti legati alle prestazioni;
- test di **sforzo**, per valutare il comportamento del componente o del sistema se sottoposto a carichi di lavoro che eccedono quelli definiti dai requisiti;
- test di **penetrazione**, per valutare la sicurezza del sistema simulando un attacco informatico contro di esso;
- test di **usabilità**, per valutare l'efficacia dell'esperienza utente.

La scelta e conseguente messa in opera di quali tipi di test coinvolgere nelle fasi di sviluppo del sistema dipendono dagli obiettivi che si desidera raggiungere o dagli aspetti più convenienti da analizzare.

2.2 Gamification

La gamification consiste nell'applicazione di elementi di game design in ambiti slegati da esso [8], con lo scopo di aumentare il coinvolgimento o influenzare il comportamento del target a cui tali ambiti si riferiscono.

Una caratterizzazione degli elementi di game design sopra citati è stata fornita da Robson et al. con il framework MDE [20], il quale distingue i concetti di meccaniche, dinamiche ed emozioni.

Le *meccaniche* sono elementi ideati dai designer dell'esperienza. Riguardano azioni che i giocatori possono intraprendere, regole e limiti del contesto di gioco, risultando quindi invariabili di giocatore in giocatore. Le *dinamiche* sono modi di approcciarsi alle meccaniche che emergono successivamente alla loro ideazione e variano in base all'esperienza del singolo utente. Sono prevedibili solo parzialmente in fase di design. Le *emozioni* sono stati interiori in cui il giocatore può trovarsi nel corso dell'esperienza. Sono l'obiettivo che il designer si pone di raggiungere attraverso la messa in atto delle meccaniche scelte. Tutti questi elementi si influenzano a vicenda, determinando l'esito dell'esperienza di gamification.

Secondo una ricerca di Figol et al. [11] i principali ambiti in cui viene fatto uso di gamification sono business, bancario, educazione e scienza, marketing e medicina. In generale si nota come – indipendentemente dalla sfera di applicazione – gli scopi a cui si punta prevalentemente siano attrarre e fidelizzare utenti, influenzarne il comportamento (ad esempio inducendoli a seguire uno stile di vita sano o a fare determinate scelte per acquisti) o porre la loro attenzione su alcuni aspetti specifici (ad esempio sul completamento del proprio profilo). Si nota inoltre l'attenzione (non da parte di tutti gli attori in gioco) sull'effettiva qualità dell'implementazione delle meccaniche di gamification. In particolare viene sottolineato come in ambito bancario un abuso sconsiderato di esse possa portare l'utente a percepire poca cura in relazione all'aspetto principale (quello finanziario) del prodotto.

Un esempio virtuoso di applicazione di tecniche di gamification è quello di Foldit [16], un gioco che propone una serie di livelli sotto forma di enigmi riguardanti ripiegamenti di proteine. Lo sforzo combinato dei giocatori ha permesso, fra le altre cose, di risolvere in pochi giorni un problema decennale riguardante la struttura di una proteina legata allo sviluppo di un virus simile all'HIV.

2.2.1 Il framework Octalysis

Octalysis è il framework di gamification – ideato da Yu-kai Chou e descritto nel suo libro "Actionable Gamification: Beyond Points, Badges and Leaderboards" [15] – a cui verrà fatto riferimento per il design dell'applicazione. Tale framework suddivide le principali meccaniche di gamification in otto gruppi (da lui definiti "core drive") in base agli aspetti psicologici e alle motivazioni che vanno a toccare per stimolare il giocatore. Segue l'elenco dei core drive con una breve descrizione per ognuno di essi:

- **"Epic Meaning and Calling"**, strettamente legato alla narrativa, avvicina le persone dando loro l'idea di star affrontando o contribuendo a qualcosa di

più grande per mezzo di uno strato narrativo che si posiziona al di sopra del sistema stesso;

- **"Development and Accomplishment"**, si basa sul senso di apprezzamento della progressione, dell'apprendimento di nuove abilità e sull'appagamento derivante dal raggiungimento di obiettivi noti;
- **"Empowerment of Creativity and Feedback"**, stuzzica la parte creativa dell'utente, fornendo le basi per influenzare a suo piacimento l'andamento del gioco (nei limiti delle regole);
- **"Ownership and Possession"**, fa leva sul desiderio di possesso e sul bisogno di prendersi cura e migliorare ciò che si è riusciti a ottenere;
- **"Social Influence and Relatedness"**, riguarda l'ambito sociale e le interazioni interpersonali che influenzano costantemente il nostro modo di approcciarci a ciò che ci circonda;
- **"Scarcity and Impatience"**, si concentra sull'attesa come amplificatore del desiderio, sull'impossibilità di ottenere qualcosa immediatamente o sulla necessità di effettuare sforzi non indifferenti per raggiungere un obiettivo;
- **"Unpredictability and Curiosity"**, sfrutta tecniche basate sul caso o sull'ignoto per stimolare la curiosità dell'utente;
- **"Loss and Avoidance"**, si basa sulla paura di perdere i progressi fatti o qualcosa che si possiede e sull'avversione a eventi negativi che possono accadere in seguito a determinate azioni.

Ognuna di queste unità contiene elementi e tecniche che possono essere sfruttati e combinati per applicare la gamification all'interno del proprio prodotto. Tuttavia, va notato come un uso inadeguato di esse o soprattutto sproporzionato in relazione alle unità di appartenenza possa comportare effetti indesiderati nei risultati ottenuti. Per questo motivo l'autore del framework effettua suddivisioni a più alto livello rispetto a quella illustrata pocanzi, che vengono raggruppate in "Left Brain" o "Right Brain" e "White Hat" o "Black Hat".

La prima suddivisione (i cui nomi non vogliono avere parvenza scientifica, ma soltanto trasmettere un'idea generale di ciò che rappresentano) differenzia le unità che si basano principalmente sul pensiero logico e analitico ("Development and Accomplishment","Ownership and Possession" e "Scarcity and Impatience", dette "Left Brain") da quelle improntate su aspetti più emotivi ("Empowerment of Creativity and Feedback", "Social Influence and Relatedness" e "Unpredictability and Curiosity", dette "Right Brain").

Si nota come le tecniche appartenenti alla prima categoria siano molto più usate delle seconde, in quanto più immediate da implementare e più rapide nel fornire risultati. Bisogna comunque sottolineare che l'abuso generale di questi aspetti potrebbe portare a stagnazione o addirittura ad allontanare potenziali utenti, motivo per cui è fondamentale bilanciare (eventualmente anche in fasi successive a quella di avvicinamento al prodotto) con meccaniche appartenenti al gruppo "Right Brain", che se applicate a dovere possono fornire all'utente motivazioni per continuare ad approcciarsi al prodotto molto più profonde di quelle toccate dal gruppo "Left Brain".

Una seconda suddivisione oppone il fronte "White Hat" ("Epic Meaning and Calling", "Development and Accomplishment", "Empowerment of Creativity and Feedback") a quello "Black Hat" ("Scarcity and Impatience", "Unpredictability and Curiosity", "Loss and Avoidance"). A differenza della prima suddivisione esposta, questa non si riferisce all'aspetto della personalità dell'utente (logico o creativo) a cui puntano le tecniche di gamification, ma alle sensazioni che suscitano queste ultime. In particolare, le unità appartenenti al gruppo "White Hat" contengono elementi che comportano soddisfazione e senso di controllo per l'utente, mentre quelle facenti parte del gruppo "Black Hat" fanno affidamento su ansia, dipendenza e simili.

In senso pratico, la principale differenza fra i due gruppi riguarda la capacità delle "Black Hat" di trasmettere un'urgenza (mancante in quelle "White Hat") capace di attirare e ammaliare più utenti, condizionandone il comportamento con risultati più repentini. Questi aspetti possono tornare utili in ambiti destinati al consumo immediato come, ad esempio, un sito di acquisti online, ma per esperienze progettate per durare nel tempo potrebbero portare a risultati indesiderati. In mancanza di altri stimoli, l'utente baserebbe il proprio legame con il prodotto unicamente sulla dipendenza sviluppata col tempo: è ragionevole pensare che prima o poi tale condizione verrà spezzata dall'accumulo di sensazioni negative o molto più probabilmente dalla scoperta di stimoli nuovi e più attraenti di quello attuale. Le tecniche "White Hat" risultano quindi necessarie per apportare un bilanciamento, in modo da suscitare nell'utente motivazioni più positive, sostenibili e a lungo termine. Bisogna comunque sottolineare che le unità "Black Hat" non vanno viste come unicamente negative: se bilanciate e sfruttate con trasparenza possono risultare strumenti efficaci di avvicinamento al proprio prodotto, possibilmente ai danni della concorrenza.

Core Drive	**Left/Right Brain**	**Black/White Hat**
Epic Meaning and Calling	-	White
Development and Accomplishment	Left	White
Empowerment of Creativity and Feedback	Right	White
Ownership and Possession	Left	-
Social Influence and Relatedness	Right	-
Scarcity and Impatience	Left	Black
Unpredictability and Curiosity	Right	Black
Loss and Avoidance	-	Black

Tabella 2.1: I Core Drive di Octalysis

2.2.2 Gamification applicata al Software Testing

Come esposto nei paragrafi precedenti, i vantaggi derivanti dall'adozione della gamification si estendono per più ambiti, inclusi quello educativo e ingegneristico. Nel primo è largamente sfruttata per coinvolgere gli studenti attraverso mezzi di insegnamento meno comuni rispetto a quelli classici, atti ad aumentare attenzione e interesse verso le materie; nel secondo (più legato a un ambiente lavorativo) gli elementi di gamification permettono di ricompensare a dovere i dipendenti (non solo economicamente), indurli a svolgere in maniera più interessante alcune attività meccaniche o poco coinvolgenti e a rafforzare il legame dipendente-azienda e dipendente-dipendente. In entrambi i casi si ipotizza un'integrazione progettata metodicamente degli elementi di gamification (le conseguenze dovute a una scorretta applicazione degli stessi sono state trattate nei paragrafi precedenti).

Per quanto concerne questo lavoro di tesi, l'attenzione è stata incentrata sull'applicazione della gamification nell'ambito dell'insegnamento del testing del software. Tra i tool e piattaforme sviluppati (per la cui stragrande maggioranza sono stati riscontrati risultati positivi) ne sono stati selezionati alcuni per analizzarne funzioni e tecniche implementate.

Il primo preso in esame è CodeDefenders [21] (Figura 2.4), in cui due giocatori si affrontano sulla base di una classe Java: uno di loro sarà l'attaccante e l'altro il difensore. Scopo dell'attaccante sarà quello di scrivere mutanti (varianti della classe in questione), mentre il difensore dovrà scrivere dei test adatti a scovarli. In caso di sospetto di mutanti equivalenti (ovvero che si comportano esattamente come la classe data, risultando quindi impossibili da rintracciare da parte dei test), il difensore può accusare l'attaccante, che dovrà rispondere scrivendo un test che possa provare che il mutante da lui scritto non sia equivalente. Il gioco si svolge a turni e i punti vengono assegnati al difensore in base ai mutanti scovati e alle accuse di mutanti equivalenti vinte, mentre all'attaccante in base al numero di mutanti che

passano i test e al numero di test di successo scritti per provare l'assenza di mutanti equivalenti in caso di accusa. In riferimento all'Octalysis sono stati riscontrati elementi di gamification provenienti dalle seguenti unità:

- "Epic Meaning and Calling": l'utilizzo di termini legati ad "attacco" e "difesa" aiuta i partecipanti ad astrarre la loro attività da una "semplice" scrittura di codice;
- "Development and Accomplishment", chiaramente legata alla presenza di punti ottenuti in base all'esito delle partite; è presente anche una sorta di classifica, la quale è tuttavia volta più al risultato di squadra che a quello personale;
- "Empowerment of Creativity and Feedback", un aspetto forse intrinseco nell'informatica, poiché la scrittura di mutanti e test è strettamente legata al modus operandi del singolo giocatore;
- "Social Influence and Relatedness", toccato dalla presenza del concetto di competizione e dalla suddivisione dei giocatori in squadre.

È interessante riportare una considerazione degli autori Rojas e Fraser, i quali fra le sfide aperte citano anche un'eventuale possibilità di astrarre il gioco dallo scrivere codice: attività (la seconda) che preclude la possibilità di approcciarsi a CodeDefenders a coloro che non hanno dimestichezza con questo mondo. Una bozza di soluzione da loro proposta si rifà fortemente alla prima unità dell'Octalysis, poiché propone di rappresentare la classe come una città, le mutazioni come degli attaccanti e i test come le truppe di difesa.

Risulta particolarmente interessante anche un framework progettato e sviluppato da Costa et al. [10] per l'insegnamento e apprendimento di test di tipo esplorativo, di cui verranno riportati i principali dettagli a seguire. Tale framework – che spicca particolarmente per l'integrazione di una componente narrativa che trae ispirazione dal gioco della caccia al tesoro e dal film "Pirati dei Caraibi" – prevede la presenza di tre tipi di partecipanti: uno "specialista" (istruttore che possiede una conoscenza completa del sistema da testare), un giudice (osservatore che prende nota delle azioni degli studenti) e dei tester (gli studenti). Dopo aver personalizzato il proprio avatar, i tester iniziano a esplorare il sistema, trovando eventuali difetti, assegnando loro un livello di priorità e generando un report dell'esito dell'attività svolta. A questo punto, ai tester viene chiesto di analizzare i report degli avversari secondo criteri prestabiliti e assegnare loro un voto con allegata una giustificazione. Lo specialista valuta le azioni riportate dal giudice e procede a premiare i tester che hanno performato meglio nelle due fasi precedentemente esposte. In riferimento all'Octalysis, le unità principali coinvolte sono le seguenti:

- "Epic Meaning and Calling", costituita principalmente dall'ambientazione piratesca, che coinvolge anche i nomi (appartenenti a pirati realmente esistiti)

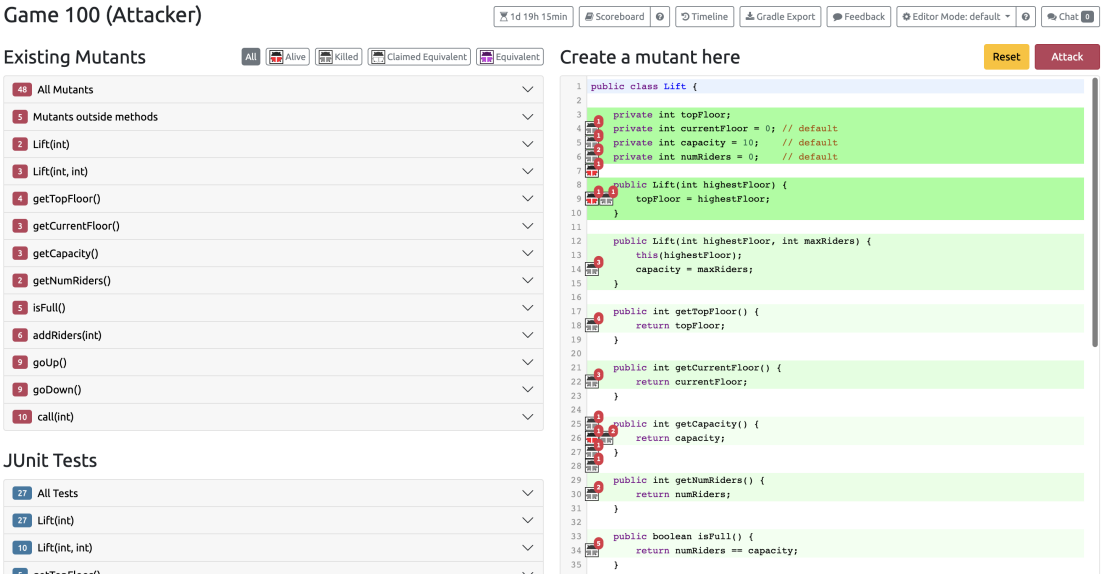

Figura 2.4: Una vista dell'interfaccia grafica del tool Code Defenders.

che vengono assegnati ai tester, i nomi delle fasi di gioco ("Trova il tesoro", "Allena il pirata", "Combatti in battaglia", ecc.) e quelli dei livelli raggiunti (che si rifanno ai nomi di alcuni personaggi di "Pirati dei Caraibi");

- "Development and Accomplishment", con la presenza di punti ottenuti in base alle prestazioni nei livelli e medaglie ottenibili in base al comportamento in classe;
- "Ownership and Possession", con avatar da personalizzare, risorse e punti da scambiare per personalizzare l'avatar;
- "Social Influence and Relatedness", data principalmente dalla presenza di una fase di conflitto fra i tester;
- "Unpredictability and Curiosity", garantita dalla possibilità di ottenere punti bonus in base al comportamento tenuto dai tester in classe.

L'interesse destato dal framework appena illustrato deriva principalmente dal suo utilizzo della prima unità principale di Octalysis: è raro, soprattutto in ambito accademico, trovare esempi simili di applicazione così forte di un determinato contesto narrativo. Nel caso del framework in questione, tale approccio ha riscontrato pareri positivi da parte dei partecipanti all'esperimento di validazione, che hanno apprezzato la coerenza del legame fra il test esplorativo e il contesto piratesco della caccia al tesoro.

Il tool Testable [17], incentrato sull'ambito dello unit testing, fa dell'unità di Octalysis relativa alla narrazione la propria meccanica principale, fornendo al giocatore un contesto narrativo completo. Nel mondo di gioco vi è Buggy, un insetto che vuole dimostrare di essere un buon programmatore come gli alieni, suoi rivali; il protagonista inizia a scrivere codice, ma a causa della sua inesperienza continua a scoprire nuovi errori in più parti del programma, finché non viene a conoscenza dell'esistenza dei test d'unità. L'utente viene quindi guidato da Buggy in una serie di schermate in cui scrivere delle funzioni (la cui correttezza verrà controllata da alcuni test predefiniti) e i relativi test. Mantenendo sempre Octalysis come riferimento, sono state riscontrate – anche in base a quanto descritto dagli autori del tool – meccaniche appartenenti alle unità di:

- "Epic Meaning and Calling": come affermato precedentemente è l'unità preponderante, data la presenza di elementi come contesto, personaggi e dialoghi;
- "Development and Accomplishment": sono presenti punti accumulabili con la progressione nei livelli;
- "Ownership and Possession", con l'integrazione di una moneta di gioco;
- "Social Influence and Relatedness", vista la possibilità di ottenere ricompense condividendo il tool su social media;
- "Unpredictability and Curiosity", per le ricompense casuali che è possibile ricevere.

Testable risulta meritevole di attenzione per più motivi. Innanzitutto per la forte implementazione di un contesto narrativo che accompagna l'utente in tutte le fasi di gioco. Rara è anche la presenza di un'interfaccia grafica apparentemente curata e in linea con la metafora di base (Figura 2.5). È interessante, infine, l'integrazione di una componente "social" derivante dalla possibilità di ottenere ricompense invitando altri utenti ad approcciarsi al tool e risulta in linea con la possibilità di accedere a Testable tramite un qualsiasi browser moderno.

Figura 2.5: Una vista dell'interfaccia grafica del tool Testable.

Capitolo 3

Design

Questo capitolo si apre con una descrizione dello stato attuale del contesto in cui Unit Brawl verrà integrato, passando successivamente alla spiegazione delle fasi del processo in seguito all'implementazione della gamification. Si chiuderà infine con un elenco e motivazione delle meccaniche di gamification adottate in fase di progettazione.

3.1 Contesto di applicazione

Il contesto a cui sarà destinata la piattaforma è quello del corso di PO del Politecnico di Torino, il cui scopo è quello di trasmettere agli studenti i concetti del paradigma di programmazione a oggetti per mezzo del linguaggio Java. In aggiunta a lezioni ed esercitazioni in aula, il corso di prevede lo svolgimento di alcune esercitazioni in laboratorio che consistono nella risoluzione di esercizi (una traccia di esercizio comune a tutti gli studenti per ogni laboratorio) mediante la scrittura di programmi Java. Queste esercitazioni sono di frequenza settimanale, per cui ogni studente può completare gli esercizi e fornire la propria soluzione entro una settimana dall'inizio dell'esercitazione. A ogni studente viene assegnato un repository personale (gestito per mezzo del software Apache Subversion) in cui caricare le proprie soluzioni ai laboratori. Al termine della scadenza del laboratorio, un server eseguirà dei test su quanto caricato dagli studenti e fornirà loro un report sull'esito tramite email. In Figura 3.1 è illustrato l'attuale processo di svolgimento dei laboratori.

Lo sviluppo di test è escluso dagli scopi di tali laboratori. A causa di ciò, si nota come gli studenti pongano scarsa attenzione ai requisiti indicati nelle tracce, consegnando quindi soluzioni che spesso presentano bug.

Inoltre, la natura facoltativa dei laboratori porta molti studenti a smettere di frequentarli già dopo poche settimane dall'inizio del corso.

Lo scopo che si pone Unit Brawl è quello di risolvere queste problematiche. La principale differenza con la situazione attuale riguarderà l'introduzione di nuovi requisiti in merito a unit test che gli studenti dovranno sviluppare parallelamente al resto del codice. Al raggiungimento della scadenza, i test di ogni studente verranno eseguiti sulle classi sviluppate dagli avversari, dando vita alla fase principale del processo gamificato.

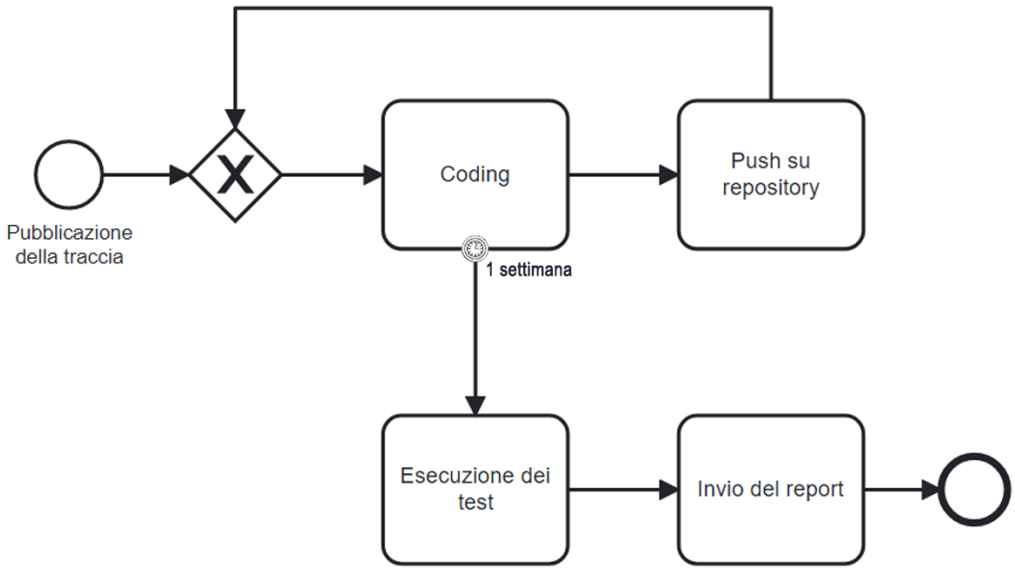

Figura 3.1: Rappresentazione in BPMN dell'attuale processo di svolgimento dei laboratori.

3.2 Processo gamificato

L'integrazione del tool nei laboratori non consisterà unicamente nell'aggiunta di meccaniche di gamification, ma prevedrà anche una modifica della struttura degli stessi. La prima differenza sostanziale riguarda il codice da consegnare: gli studenti non si occuperanno solo di scrivere metodi (il linguaggio da usare – Java – resterà lo stesso), ma dovranno anche sviluppare unit test avvalendosi del framework JUnit. Inoltre, i laboratori passeranno da una frequenza (e quindi durata) settimanale a una bisettimanale e si opterà per la piattaforma GitLab per la gestione dei repository. Infine, sarà disponibile per gli studenti una piattaforma web alla quale potranno accedere per eseguire varie azioni che verranno esplicate a seguire.

Come mostrato in Figura 3.2, il nuovo processo di svolgimento di un singolo laboratorio è composto da due macro-processi: quello di scrittura del codice e quello di valutazione al termine della scadenza del laboratorio.

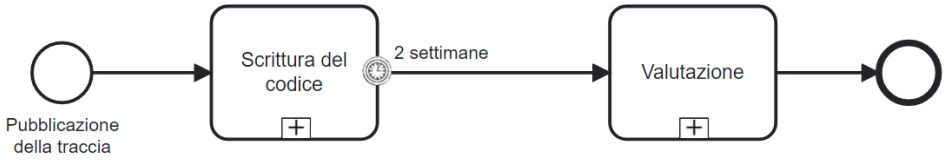

Figura 3.2: Rappresentazione in BPMN ad alto livello del nuovo processo di svolgimento dei laboratori.

La prima fase (Figura 3.3) ha inizio con la pubblicazione della traccia del laboratorio, che descrive l'applicazione da sviluppare con le classi e i metodi (le cui firme devono essere esattamente le stesse riportate all'interno della traccia): gli studenti avranno due settimane per implementare una soluzione che rispetti i requisiti richiesti. Inoltre, sarà necessario implementare anche degli unit test (il cui numero massimo varierà di laboratorio in laboratorio) che agiscano su metodi a piacimento appartenenti alle classi sviluppate. I test verranno eseguiti:

- sulla propria classe, in modo da ricavare informazioni utili a verificare la percentuale di raggiungimento di determinati obiettivi;
- a fine laboratorio sulle classi degli altri studenti, in modo da stilare una classifica in base ai punteggi ottenuti.

Ogni studente potrà eseguire una fork di un repository GitLab con una struttura predefinita in cui poter caricare la propria soluzione entro la scadenza del laboratorio. In questa fase, gli studenti potranno eseguire una push del proprio codice un numero illimitato di volte. Ogni push darà inizio a una pipeline (implementata tramite gli strumenti di Continuous Integration di GitLab) per verificare che le seguenti condizioni vengano rispettate:

- il codice riguardante i test non dà errori in fase di compilazione;

- il numero di test non eccede il massimo consentito per quel determinato laboratorio;

- i test non falliscono su una soluzione ideale preparata in precedenza (idealmente inaccessibile agli studenti).

Se almeno uno dei punti di verifica non viene rispettato, verrà generato un report accessibile tramite l'interfaccia di GitLab in modo da metterlo al corrente dei dettagli. Altrimenti, si procederà con l'esecuzione dei test sulla propria soluzione in modo da verificare le percentuali di completamento di determinati obiettivi e aggiornare una sezione specifica della piattaforma web. Ogni obiettivo raggiunto verrà effettivamente confermato tale solo successivamente alla conclusione del laboratorio, in modo da effettuare ogni considerazione sulla soluzione finale consegnata dallo studente.

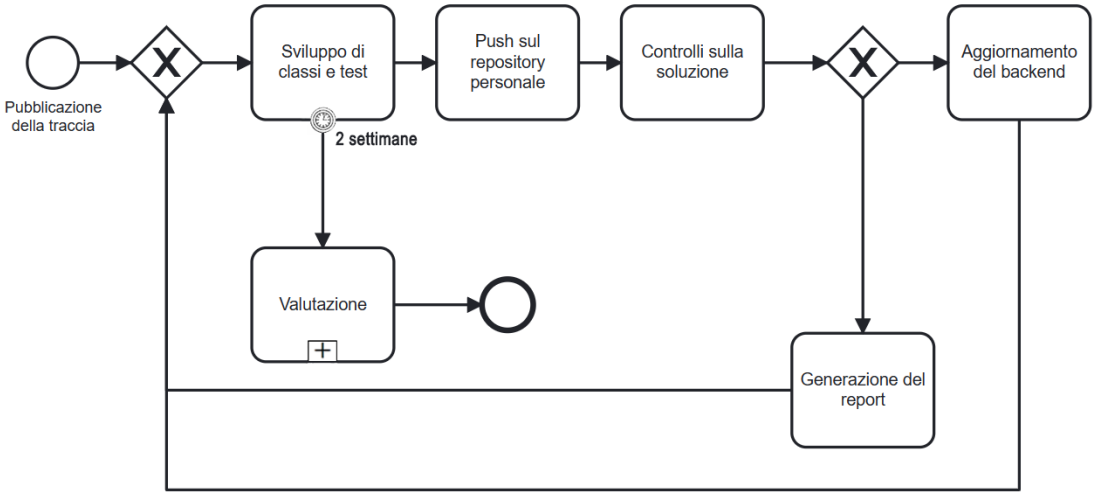

Figura 3.3: Dettaglio della rappresentazione in BPMN della fase di scrittura del codice.

Raggiunta la scadenza del laboratorio si passa alla seconda fase (Figura 3.5), quella di valutazione delle soluzioni e di aggiornamento delle informazioni mostrate dall'applicazione web. Innanzitutto, un server provvederà a effettuare una clone di ogni repository, per portare avanti inizialmente le stesse verifiche che nella fase precedente vengono fatte a ogni push (non essendo, in quella fase, bloccanti). Le soluzioni che superano questo passaggio ottengono 1 punto e passano al calcolo del punteggio finale: ogni studente otterrà 1 punto per ogni test avversario che avrà successo quando eseguito sulla propria classe (non rilevando quindi la presenza di bug) e 1 punto per ogni esito negativo che i propri test otterranno in seguito

all'esecuzione su classi degli altri studenti (ovvero per ogni bug rilevato sulle classi avversarie).

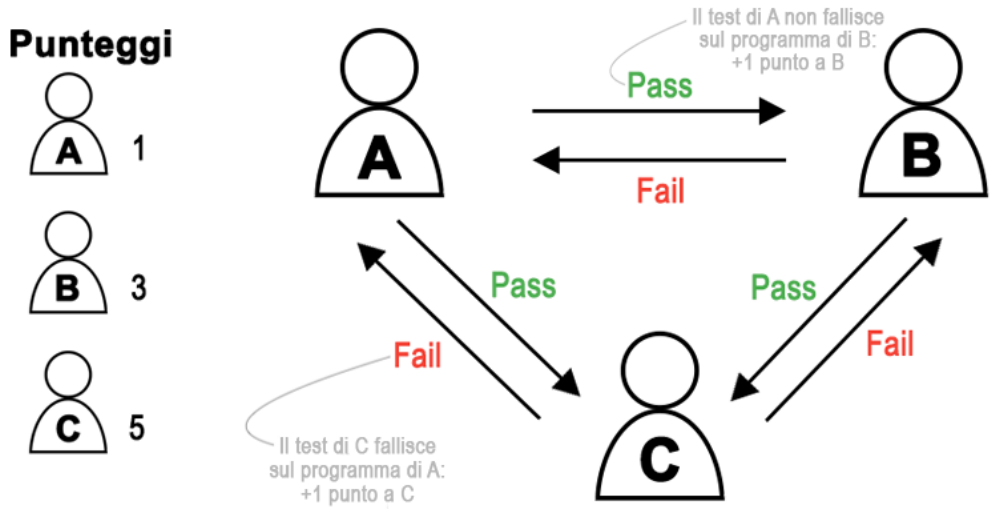

Figura 3.4: Esempio di un'ipotetica sessione che coinvolge tre giocatori.

A titolo esemplificativo, si supponga che i giocatori A, B e C sviluppino un test ciascuno. Figura 3.4 mostra l'esito della fase descritta pocanzi: alla fine della sessione, il giocatore C otterrà 5 punti (1 per aver superato i test di validazione, trova 2 bug sul codice avversario e 2 test avversari non falliscono sul proprio); similmente, B otterrà 3 punti e A 1 punto.

Il processo si conclude con l'aggiornamento delle informazioni (classifiche, obiettivi, ecc.) sull'applicazione web.

3.3 Meccaniche di gamification adottate

Di seguito verranno elencate e giustificate le meccaniche di gamification prese in considerazione. Le scelte sono state dettate tenendo conto non solo del puro concetto di gamification, ma soprattutto della sua applicazione in un contesto – quello didattico – il cui fine ultimo dovrebbe essere quello di avvicinare e invogliare lo studente all'apprendimento delle tematiche proposte (il testing, nello specifico). Fra i "motori" di avvicinamento attorno a cui ruotano le tecniche scelte vi è quello della competizione (che assume un ruolo preponderante), affiancato da meccaniche sociali, di miglioramento e di espressione di sé. Si è preferito evitare di basare tutto il sistema sulla sola competizione, poiché avrebbe potuto allontanare studenti che preferiscono approcciarsi allo studio con ottiche differenti.

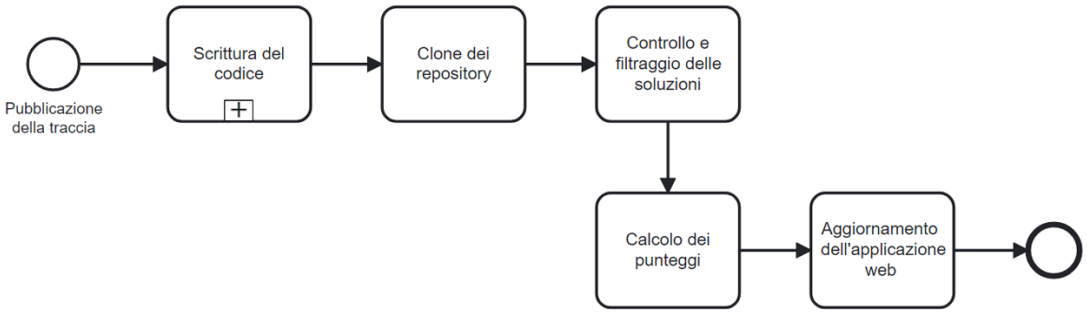

Figura 3.5: Dettaglio della rappresentazione in BPMN della fase di verifica e conclusione del laboratorio.

3.3.1 Classifiche

Lo scopo primario delle classifiche è dare un volto alla competizione, mostrando l'esito degli sforzi dei giocatori in relazione a quelli effettuati dagli avversari. Tuttavia, se da un lato possono essere strumenti di soddisfazione per i giocatori più performanti, dall'altro una loro implementazione inaccurata potrebbe portare a effetti collaterali non indifferenti. Sono state identificate le seguenti situazioni problematiche:

1. una classifica visibile da tutti e completa potrebbe demotivare i giocatori classificati in basso;

2. l'applicazione della gamification a un singolo laboratorio porterebbe alla produzione di una sola classifica, precludendo agli studenti la possibilità di imparare dai propri errori e migliorare in una futura nuova manche del gioco.

Queste considerazioni possono verosimilmente valere in svariati contesti in cui viene fatto uso di classifiche, ma assumono un peso molto maggiore in ambito didattico. La problematica (1) potrebbe demotivare uno studente classificato in basso, inducendolo erroneamente a ritenersi inferiore agli altri giocatori o a pensare di aver perso il suo tempo. La problematica (2) impedirebbe allo studente di continuare a studiare per migliorare e raggiungere traguardi sempre più alti, etichettando una classifica bassa come un punto di arrivo impossibile da superare.

Il punto (2) è di facile risoluzione, estendendo l'uso delle classifiche a più di un laboratorio. Il punto (1) è invece più complesso da risolvere e richiede di

applicare un'implementazione diversa dalla classica classifica "completa". Nel farlo si è scelto come filo conduttore il concetto di "urgent optimism", per cui l'utente deve percepire di avere sempre una possibilità di successo non indifferente, agendo nell'immediato. Tenendo a mente questo concetto, si è deciso innanzitutto di separare le classifiche dei laboratori da quella globale. In base ai punti ottenuti nel singolo laboratorio si stila la classifica legata ad esso, le cui posizioni saranno la base per l'assegnazione dei punteggi che i giocatori potranno accumulare nel corso dei laboratori e dai quali dipenderà la classifica generale. In tal modo si evita che risultati poco equilibrati derivanti da un determinato laboratorio possano sbilanciare la classifica generale, inficiando sul grado di coinvolgimento degli utenti per le sessioni successive a causa di eventuali distacchi irrecuperabili fra le prime e le ultime posizioni. La formula adottata per l'assegnazione del punteggio in base alla posizione è la seguente:

$$2000 - 5 * (position - 1)$$

Nella formula, *position* è la posizione raggiunta dal giocatore nel laboratorio. Un punteggio così calcolato vedrà assegnati al primo classificato 2000 punti, andando a scendere nelle posizioni a intervalli di 5 punti. Una configurazione del genere permette l'assegnazione di punteggi a 400 giocatori, che si stima possano essere superiori al numero di studenti che prenderanno parte ai laboratori.

Passando alla tipologia di classifiche, la scelta è ricaduta sulla seguente implementazione:

- le prime 10 posizioni vengono mostrate pubblicamente;
- se un giocatore non rientra nelle prime 10 posizioni, può consultare privatamente la sezione di classifica in cui si trova e i punti ottenuti.

Indipendentemente dall'implementazione, la classifica è una delle meccaniche più utili e sfruttate in tool che si basano sulla competizione. Il tool Gamekins [22] implementa efficacemente l'elemento della classifica, mostrando per ogni partecipante i suoi avatar e nome, il punteggio ottenuto, ma anche il numero di sfide superate e obiettivi raggiunti, in modo da rinforzare la componente competitiva. Rincon-Flores et al. hanno effettuato una scelta simile per la classifica del tool Gamit! [19], mostrando al suo interno avatar, obiettivi raggiunti e punti. I risultati — considerati positivi in relazione agli obiettivi prefissati — del loro esperimento dimostrano come la maggior parte degli utenti consulti settimanalmente la classifica in cerca di informazioni.

3.3.2 Punti

All'interno del sistema sono presenti due tipi di punti: i punti di stato e i punti scambiabili.

I primi riguardano l'esito del singolo laboratorio e non vengono cumulati da un laboratorio all'altro. Questa tipologia di punti permette di quantificare l'esito della partita, dando una misura della bontà della propria soluzione e di distanza dagli altri giocatori aggiuntiva rispetto a quella rappresentata dalla classifica.

I punti scambiabili, invece, allontanano lo sguardo dal concetto di competizione per volgerlo al "motore" della personalizzazione: i punti di stato ottenuti nel corso dei laboratori vengono convertiti in punti scambiabili (con un rapporto di 10:1) che possono essere accumulati e spesi per sbloccare nuove personalizzazioni per il proprio avatar.

Dando uno sguardo ai tool esistenti, l'utilizzo dei punti è strettamente legato alla presenza di una classifica. Meritevole di menzione è il gioco Code Defenders [21], in cui i punteggi dipendono dal ruolo del giocatore ("attaccante" o "difensore") e da dinamiche opzionali (la presenza di test equivalenti).

3.3.3 Avatar e personalizzazione

La scelta di aggiungere l'elemento degli avatar [1] all'interno del progetto è derivata da una considerazione fondamentale: non tutti gli studenti potrebbero trovare nella competizione una motivazione adeguata alla prosecuzione dei laboratori. L'aggiunta degli avatar riesce così a coprire più aspetti che potrebbero avvicinare costantemente i giocatori:

- lo stimolo della creatività per mezzo della personalizzazione del proprio avatar;
- il desiderio di possesso relativo a elementi di personalizzazione rari o costosi;
- l'espressione di sé agli occhi degli altri, mostrando il proprio avatar agli altri giocatori.

Al primo ingresso sulla piattaforma, l'utente riceve un avatar di default dall'aspetto quanto più generico possibile, in modo che ognuno possa sentirlo proprio indipendentemente dalle caratteristiche personali o culturali.

In letteratura sono stati identificati due utilizzi degli avatar degni di nota. Come accennato in un sottoparagrafo precedente, in Gamekins [22] è presente una classifica che mostra, fra le altre cose, anche l'avatar del giocatore. Quest'ultimo può scegliere tra 50 avatar diversi, avendo quindi a disposizione un elevato grado di personalizzazione dell'esperienza. Rincon-Flores et al. hanno invece giustificato la presenza degli avatar in Gamit! [19] come un modo per ridurre l'esposizione del giocatore per far pesare di meno il contraccolpo derivante da eventuali risultati insoddisfacenti.

[1] Le immagini degli avatar sono state generate tramite il sito getavataaars.com.

3.3.4 Badge e obiettivi

I badge (Tabella 3.1) sono etichette che possono essere sbloccate al raggiungimento di determinati obiettivi significativi e che comportano l'acquisizione di punti da aggiungere a quelli accumulati in relazione alla classifica globale. La loro integrazione fornisce scopi che non sono direttamente legati alla competizione, ma che possono risultare parimenti utili ai fini didattici e soddisfacenti per i giocatori.

Nome dell'obiettivo	**Descrizione**	**Punti**
Welcome	Join one lab	10
Nice to see you again	Join three labs	25
You are our constant	Join five labs	70
First steps	Get into the top 100 of one lab	10
Not-so-first steps	Get into the top 100 of three labs	30
The important thing is to participate	Get into the top 100 of five labs	50
Next level	Get into the top 50 of one lab	50
Rock solid	Get into the top 50 of three labs	100
High performance proved	Get into the top 50 of five labs	200
Quantum leap	Get into the top 20 of one lab	100
You are the best	Get into the top 20 of three labs	250
On top of the world	Get into the top 20 of five labs	500
Coverage Apprentice	Reach 50% coverage on your solution	20
Coverage Expert	Reach 70% coverage on your solution	60
Coverage Master	Reach 85% coverage on your solution	150
Junior collector	Collect four avatars	40
Richness detected	Collect ten avatars	50
Uncle $crooge	Collect twenty avatars	100

Tabella 3.1: Obiettivi implementati

Alcuni obiettivi hanno lo scopo di spingere l'utente a esplorare i vari aspetti della piattaforma (come l'acquisto di avatar), altri a dare ulteriori motivazioni per mantenere costante la frequenza di partecipazione ai laboratori, altri ancora sono legati strettamente allo svolgimento dei singoli laboratori (ad esempio al risultato dei propri test eseguiti sul proprio programma).

Se il progresso relativo al raggiungimento di tali tipi di obiettivi può essere mostrato in maniera sincrona, per assegnare effettivamente i badge ai giocatori sarà necessario attendere il termine del laboratorio. Così facendo, se alcuni giocatori scriveranno programmi più semplici con lo scopo di raggiungere facilmente determinati obiettivi, verranno comunque fortemente penalizzati dall'esito della componente multigiocatore, dal momento che verosimilmente molti test avversari falliranno su tali programmi.

Un'implementazione simile a quella appena descritta è stata riscontrata in Gamekins [22], in cui esistono degli obiettivi che possono essere raggiunti in maniera indipendente dalle altre sfide. Il numero di obiettivi raggiunti viene mostrato all'interno della classifica.

3.3.5 Progress bar

A ogni azione dell'utente deve corrispondere un feedback da parte del sistema e le progress bar sono strumenti utili in tal senso, dal momento che permettono al giocatore di ottenere un riscontro in base alla soluzione consegnata anche prima della scadenza del laboratorio.

All'interno dell'applicazione è stata associata una progress bar a ogni obiettivo, in modo da dare un'indicazione chiara e in tempo reale allo studente dei progressi mancanti per ottenere un determinato badge.

3.3.6 Scarsità del tempo

La natura periodica dei laboratori si lega perfettamente al concetto di scarsità del tempo: il giocatore non può procrastinare eccessivamente le azioni da compiere, poiché queste potranno portarlo a raggiungere uno stato di vittoria solo se compiute entro determinati limiti temporali.

Tale concetto viene implementato nell'interfaccia dell'applicazione web da un componente che mostra inizialmente la data di scadenza del laboratorio e che si tramuta in un timer inglobato da una progress bar quando questa si avvicina. Non si è optato per un timer costantemente presente poiché indurrebbe nel giocatore uno stato d'ansia dannoso vista la durata non indifferente di ogni singolo laboratorio. Tale stato di tensione può aiutare il giocatore a dare il meglio di sé solo se limitato a un frangente di tempo relativamente breve, motivo per cui l'inserimento del timer nel periodo immediatamente antecedente la scadenza del laboratorio.

Tali considerazioni vengono riportate anche da Elbaum et. al [9]: alcuni studenti scelti per testare il tool Bug Hunt hanno affermato di aver provato frustrazione a causa di un timer presente nella prima sessione del gioco.

Meccanica	**Core Drive**	**Left/ Right Brain**	**Scopo**
Classifiche	Development and Accomplishment	Left	Dare un volto alla competizione
Status points	Development and Accomplishment	Left	Quantificare i risultati
Exchangeable Points	Ownership and Possession	Left	Sblocco di personalizzazioni
Avatar	Ownership and Possession	Left	Espressione di sé
Obiettivi	Development and Accomplishment	Left	Aggiungere obiettivi personali
Progress bar	Development and Accomplishment	Left	Dare un'idea di progresso
Scarsità del tempo	Scarcity and Impatience	Left	Stimolare i giocatori
Scrittura del codice	Empowerment of Creativity and Feedback	Right	Lasciare al giocatore le redini della propria partita
Feedback in tempo reale	Empowerment of Creativity and Feedback	Right	Dare un riscontro in tempo reale delle azioni del giocatore

Tabella 3.2: Meccaniche di gamification implementate

Capitolo 4

Implementazione

In questo capitolo verrà illustrata l'implementazione dell'applicazione che metterà in pratica quanto descritto nei capitoli precedenti. Si inizierà presentando e motivando le scelte architetturali, le tecnologie adottate e le comunicazioni fra i vari componenti del sistema. Si procederà poi con una descrizione dei casi d'uso riguardanti le possibili interazioni fra utenti e sistema, i quali costituiranno la base per i due passaggi successivi: la creazione di wireframe (con particolare attenzione all'aspetto dell'esperienza utente) e l'effettiva implementazione dell'applicazione finale.

4.1 L'architettura

Il sistema (Figura 4.1) è caratterizzato dalle interazioni fra quattro componenti principali:

- un **backend**, punto centrale delle comunicazioni fra i vari componenti, contiene al suo interno la logica di business e gestisce l'accesso ai dati del sistema;
- i **meccanismi di continuous integration offerti da GitLab**, con lo scopo di effettuare test sulla soluzione sottomessa che diano feedback rapidi all'utente senza aumentare il carico di lavoro del server;
- due **frontend web** che forniscono agli utenti (studenti o amministratori) punti di accesso per svolgere le azioni di loro interesse.

4.1.1 Backend

Il backend è stato implementato per mezzo del framework Express, basato su Node.js, un ambiente di esecuzione di codice JavaScript asincrono, open source e orientato agli eventi. La leggerezza e l'immediatezza nella configurazione di

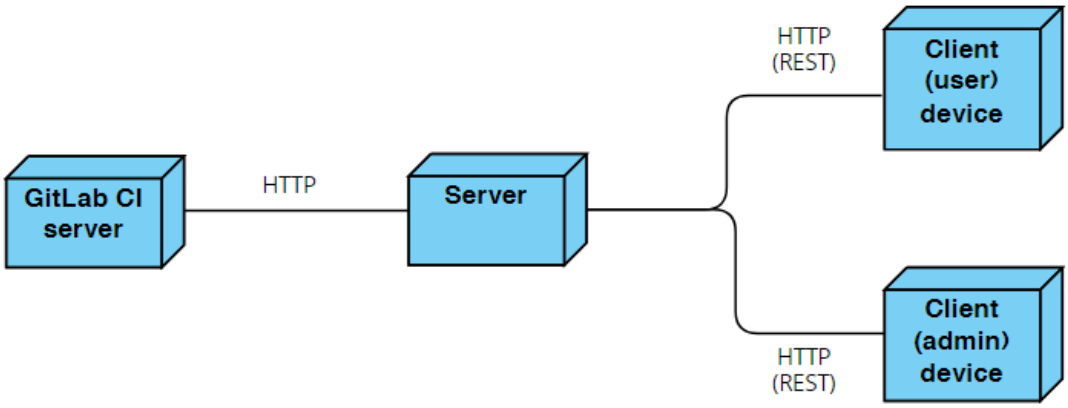

Figura 4.1: Rappresentazione del sistema per mezzo di un Deployment Diagram UML.

Node.js sono alcune delle caratteristiche risultate fondamentali nella scelta di quale tecnologia adottare per la messa in opera del server, tenendo conto della natura del progetto e delle capacità di calcolo a disposizione. Inoltre, la natura intrinsecamente non bloccante delle richieste di I\O aiuta ulteriormente a migliorare le prestazioni del server in relazione all'invio di dati ai client web.

Per quanto riguarda Express, il framework è stato scelto – oltre che per la sua flessibilità e leggerezza – in quanto mette a disposizione strumenti di routing e middleware utili a supportare lo sviluppo delle funzionalità richieste.

SQLite è stato scelto come database engine per via della sua leggerezza e delle funzionalità offerte.

4.1.2 GitLab Continuous Integration

L'integrazione delle funzionalità di Continuous Integration (CI) di GitLab all'interno del progetto ha come obiettivo principale quello di dare primi feedback all'utente (circa la validità dei test scritti e l'avanzamento nel completamento degli obiettivi) cercando di spostare quanto più calcolo possibile dal server agli ambienti di esecuzione offerti da GitLab.

Le funzionalità di Continuous Integration (CI) di GitLab si basano sulla presenza di un file .gitlab-ci.yml nella root del repository. All'interno di questo file è possibile definire gli script da eseguire in maniera automatica dopo l'avvenimento di determinati eventi di interesse. Tale pipeline di esecuzione potrà essere eseguita in ambienti denominati GitLab Runners, offerti in remoto dallo stesso GitLab o installati personalmente. Gli script sono raggruppati in job, che all'interno della pipeline di esecuzione possono avvenire in sequenza o in parallelo gli uni con gli

altri. Inoltre, è possibile eseguire gli script all'interno di container Docker in modo da sfruttare le funzionalità offerte dal container scelto. È possibile controllare l'esito dei vari stadi della pipeline nell'apposita sezione "Pipelines". Le pipeline definite nel file di CI possono essere lanciate in seguito a eventi di diverso tipo: le branch pipeline vengono eseguite ad ogni push sul branch e hanno accesso alle variabili di progetto protette, mentre le merge request pipeline vengono eseguite nel momento in cui si crea una nuova merge request e non hanno accesso alle variabili di progetto protette. In particolare, le merge request pipeline possono essere avviate su progetti nati da un fork: verranno eseguite sul contenuto del progetto origine della merge request (quello nato da fork) e non sul progetto destinazione (quello originale).

Scenario	**Descrizione**	**Pro**	**Contro**
A: branch pipeline e calcolo sul server.	Sfruttare la CI di tipo branch pipeline per aggiornare il server a ogni push dell'utente. In seguito alla ricezione dell'avviso, il server clona la repository dell'utente, effettua i primi controlli di validità e aggiorna il database.	Nessuna esposizione di informazioni sensibili dal momento che tutti i controlli avvengono all'interno del server.	Il server esegue tutto il calcolo.
B-1: branch pipeline e calcolo su GitLab.	La pipeline di CI viene avviata ad ogni push per eseguire i primi controlli riguardanti la validità dei test scritti dal giocatore, il cui esito potrà essere controllato nell'apposita sezione dell'interfaccia di GitLab. Per effettuare tali test sarà necessaria la presenza di una soluzione ideale su un repository remoto, il cui link potrà essere messo a disposizione della pipeline di CI sotto forma di variabile di progetto nascosta.	Il server effettua i calcoli solo alla fine del laboratorio e senza bisogno di rapidità nel fornire i risultati; l'utente può ottenere primi feedback direttamente su GitLab; le branch pipeline permettono di accedere alle variabili di progetto nascoste.	Non sarà possibile aggiornare l'utente in tempo reale sul completamento dei propri obiettivi; il valore delle variabili nascoste può essere facilmente scoperto manomettendo il file di CI.
B-2: branch pipeline, calcolo su GitLab e aggiornamento del server.	La CI viene usata non solo per effettuare i test bloccanti, ma anche quelli relativi al completamento degli obiettivi. L'esito di tali test verrà inviato al server, che potrà salvarlo e renderlo disponibile all'utente attraverso il frontend.	L'utente può ricevere in tempo reale feedback non solo sulla validità dei test, ma anche sul completamento degli obiettivi.	Un coinvolgimento così frequente del server potrebbe essere gravoso a livello di prestazioni; restano presenti i problemi relativi alla visibilità delle variabili segrete.

Scenario	**Descrizione**	**Pro**	**Contro**
C-1: branch pipeline e merge request pipeline (una sola merge)	Le branch pipeline vengono sfruttate per i controlli sulla validità dei test (il cui esito sarà visualizzabile sull'interfaccia di GitLab) e al termine della fase di scrittura del codice l'utente potrà effettuare una merge request e scatenare l'esecuzione della merge request pipeline, che indica al server di clonare la repository, effettuare i calcoli sulla percentuale di raggiungimento degli obiettivi e mostrarli attraverso il frontend. L'utente potrà fare una sola merge request in tutta la durata del laboratorio.	In fase di scrittura dei test il server non viene coinvolto a ogni push di ogni utente, ma a ogni merge request (effettuabile da ogni utente al più una volta).	I feedback sulla percentuale di completamento degli obiettivi arrivano successivamente alla consegna della soluzione per mezzo della merge request: l'informazione diventa quindi inutile ai fini del miglioramento personale non essendoci più possibilità di modificare la soluzione sottomessa; le merge request pipeline non hanno accesso alle variabili di progetto nascoste.
C-2: branch pipeline e merge request pipeline (più merge)	L'utente può effettuare più merge request. Lo scopo delle branch pipeline resta invariato rispetto al caso precedente, mentre le merge request pipeline avviano l'aggiornamento del completamento degli obiettivi e inviano le informazioni al server.	L'utente può provare a migliorare nell'arco della durata dello stesso laboratorio; il server torna ad essere coinvolto durante la fase di scrittura del codice, ma ipoteticamente in frequenza minore rispetto ai casi precedenti.	Resta irrisolto il problema dell'impossibilità di accedere alle variabili di progetto nascoste; l'alternanza di push e merge request in base al fine che si vuole raggiungere potrebbe creare confusione nel giocatore a livello di regole di gioco.

Tabella 4.1: Possibili scenari di implementazione della CI offerta da GitLab

Considerati gli aspetti tecnici, sono stati definiti vari scenari di integrazione (Tabella 4.1.2 e Tabella 4.1), alcuni dei quali volutamente limitati in modo da identificare i fronti di miglioramento. Come premessa per ogni scenario bisogna specificare che per ogni laboratorio l'utente effettuerà una fork di un progetto di partenza e lo renderà pubblico in modo da essere clonato successivamente dal server.

Per quanto riguarda il modo in cui il server possa associare il giocatore alla soluzione caricata su GitLab, sono state considerate due soluzioni:

- **S-1**: all'avvio del laboratorio, il giocatore effettua una fork del progetto e ne comunica il link al server tramite un campo apposito nella web app (tale azione sarebbe paragonabile a una richiesta di partecipazione del giocatore al laboratorio);
- **S-2**: si associa un id del giocatore a ogni richiesta inviata al server tramite il file di CI di GitLab.

La soluzione S-2, tuttavia, presenta alcuni problemi. Innanzitutto, il server dovrebbe riuscire ad associare l'id GitLab dell'utente ricevuto nelle richieste al proprio id riguardante quell'utente: per risolvere si potrebbe obbligare l'utente a registrarsi al sistema con uno username o indirizzo email che sia uguale a quello GitLab, ma la soluzione non è esente da errori in quanto dipende da input inseriti dall'utente stesso. Inoltre, l'id GitLab verrebbe inviato al server tramite uno script nel file di configurazione della CI, che risulta anche qui essere un facile punto di manomissione: un utente malevolo potrebbe facilmente inviare al server informazioni false su altri utenti semplicemente cambiando il valore associato all'id nel file.

Per quanto riguarda i vari scenari illustrati, si è deciso di scartare con sicurezza lo scenario A in quanto ideato in maniera volutamente limitata in modo da prenderlo come punto di partenza per quelli successivi. Se si desidera mantenere la possibilità per l'utente di consultare l'avanzamento in merito al completamento degli obiettivi, è necessario scartare B-1 e C-1 (quest'ultimo in quanto permetterebbe all'utente di consultare i risultati in questione solo dopo aver consegnato la soluzione, senza possibilità di modificarla e risultando quindi equivalente a un aggiornamento degli stessi alla fine del laboratorio). Degli scenari restanti, B-2 risulta quello più diretto da implementare e più facile da comprendere per il giocatore, mentre C-2 quello più favorevole (almeno apparentemente) in ambito di prestazioni. In entrambi i casi il coinvolgimento del server nella fase di svolgimento del laboratorio non richiederebbe particolari calcoli da parte sua (se non quelli di validazione delle richieste), poiché ogni richiesta in questa fase consisterebbe in nient'altro che un'aggiunta di informazioni nel database. La scelta di sfruttare Node.js per l'implementazione del server potrebbe essere favorevole a questo scenario grazie alla natura non bloccante dello stesso nell'effettuare richieste di I/O.

Preso atto di queste considerazioni, la scelta finale è ricaduta sullo scenario B-2, identificato come un punto di incontro ragionevole fra facilità d'uso e prestazioni.

Per quanto riguarda l'associazione utente-repository, la soluzione ideale richiede un'implementazione combinata di S-1 e S-2. Per inviare dati al server nel corso del laboratorio si dovrebbe optare necessariamente per S-2, poiché altrimenti non ci sarebbe altro modo di associare i dati sul completamento degli obiettivi all'utente che li sta aggiornando. Viste però le debolezze di tale soluzione, il server dovrebbe effettuare i controlli finali sulla base di dati più affidabili, motivo per cui si ritiene necessario integrare anche S-1 come azione da effettuare per partecipare al laboratorio: in tal modo il server avrà il link tramite cui clonare il repository a fine laboratorio ed eseguire adeguatamente tutti i controlli del caso.

```
stages:
  - all
  - send
all:
  stage: all
  image: maven:latest
  script:
    #clone ideal solution
    - git clone https://gitlab.com/admin/test-ideal-solution.git
    #formality test
    - cp test-ideal-solution/lab/src/test/java/FormalityTest.java
   lab/src/test/java
    - cd lab
    - mvn test -Dtest="FormalityTest"
    - cd ..
    - rm lab/src/test/java/FormalityTest.java
    - rm test-ideal-solution/lab/src/test/java/FormalityTest.java
    #test on ideal solution
    - cp lab/src/test//java/TestClass.java test-ideal-solution/lab
   /src/test/java
    - cd test-ideal-solution/lab
    - mvn package
    - cd ../..
    #test on own solution and report generation
    - cd lab
    - mvn package
    - cd ../..
  artifacts:
    paths:
      - lab/target/site/jacoco/jacoco.xml
send:
  stage: send
  variables:
    HEADER1: "content-type: text/plain"
    HEADER2: "accept: text/plain"
  script:
```

```
35      #send coverage report to server
36      - curl -s -H "$HEADER1" -H "$HEADER2" -d "$(cat lab/target/
    site/jacoco/jacoco.xml)"  $SERVER_URL/gitlab/coverage/
    $GITLAB_USER_LOGIN
```

Listing 4.1: Contenuto del file .gitlab-ci.yml

4.1.3 Frontend web

Il sistema vede la presenza di due frontend web, che si interfacciano rispettivamente con giocatori e amministratori. Per l'implementazione del frontend si è optato per la libreria React, basata su JavaScript e sulla sintassi JSX (nel caso specifico del progetto). Fra i vantaggi che hanno guidato la scelta vi sono (oltre a performance, flessibilità e immediatezza d'uso) la possibilità di costruire viste in maniera dichiarativa (senza occuparsi del rendering in maniera diretta) e l'uso e riutilizzo di componenti che contengono al loro interno e permettono di definire la logica per la gestione del loro stato. Inoltre è stato fatto largo uso di componenti facenti parte del framework React Bootstrap, in modo da velocizzare lo sviluppo dando al contempo uno stile uniforme all'interfaccia.

4.2 Requisiti e casi d'uso

L'esperienza utente è stata ideata a partire dalle possibili interazioni fra utente e sistema. Innanzitutto, sono state identificate due tipologie di utenti: il giocatore e l'amministratore. Conseguentemente sono stati ricavati i requisiti funzionali dell'applicazione, modellati successivamente in casi d'uso. Seguono i principali requisiti che coinvolgono l'utente di tipo giocatore:

- registrarsi al sistema (scegliendo un nome utente visibile globalmente e una password) e successivamente effettuare il login;
- implementare e caricare la propria soluzione al laboratorio;
- consultare la classifica generale;
- ottenere informazioni sul laboratorio in corso (traccia, scadenza, ecc.) e su quelli passati;
- ottenere informazioni sul proprio stato (risultati ottenuti ai precedenti laboratori, monete a disposizione, percentuale di completamento degli obiettivi, ecc.);
- acquistare nuovi avatar e visualizzare quelli in proprio possesso;

- impostare un avatar fra quelli in proprio possesso come immagine del profilo;
- informarsi sul funzionamento della piattaforma e sulle regole di gioco.

Tali requisiti (rappresentati come casi d'uso ad alto livello in Figura 4.2) possono essere suddivisi in sette aree tematiche: registrazione/login, coding, classifica, laboratori, acquisti, profilo e istruzioni.

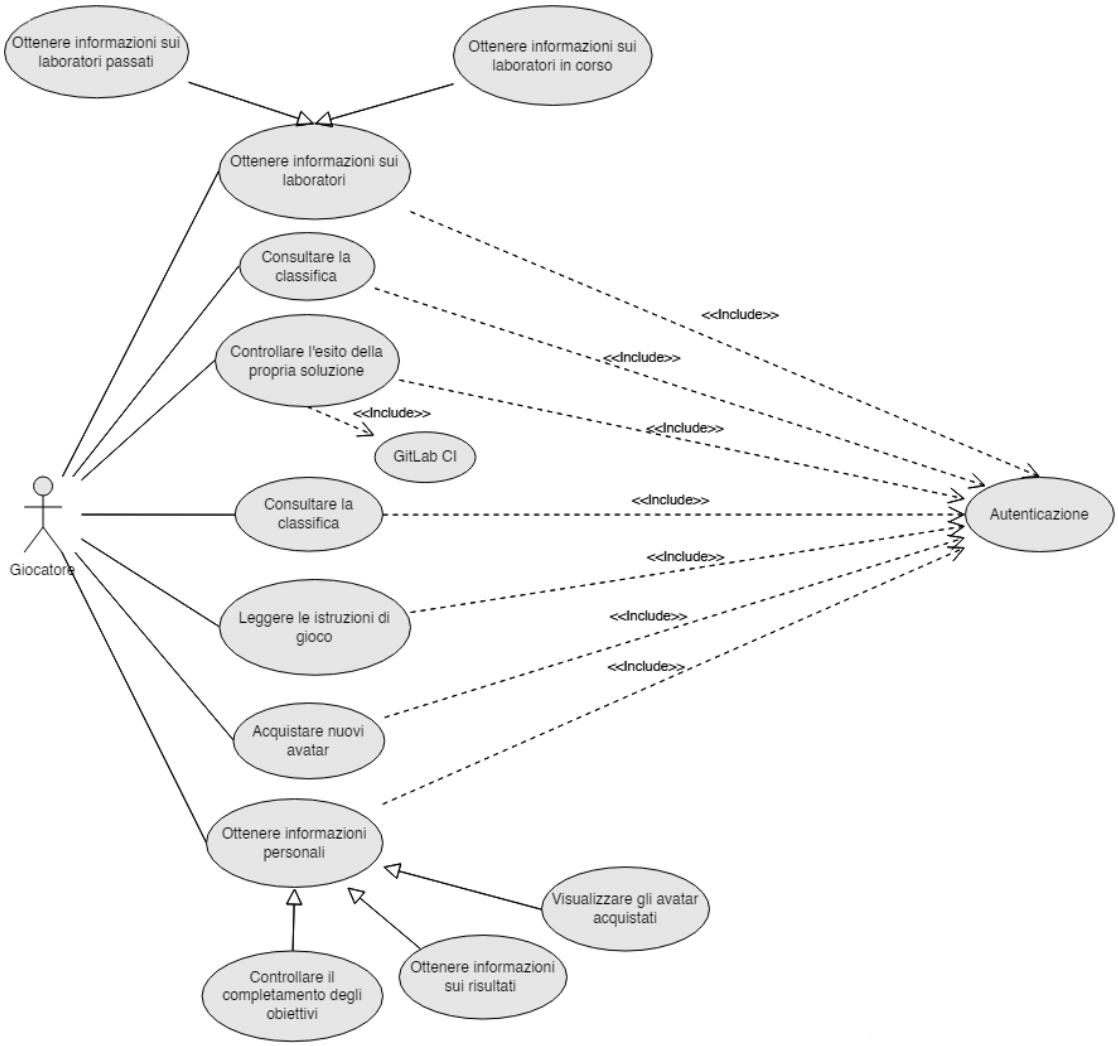

Figura 4.2: Rappresentazione per mezzo di uno Use Case Diagram UML dei principali casi d'uso che coinvolgono il giocatore.

Per quanto riguarda l'amministratore, i principali requisiti contemplati sono:

- effettuare il login alla piattaforma;

- avviare un nuovo laboratorio o terminarne uno in corso;
- ottenere informazioni sul laboratorio in corso;
- ottenere report su aspetti riguardanti giocatori e laboratori;
- inserire, modificare o eliminare elementi di gioco.

Questi ultimi possono essere raggruppati in: autenticazione, gestione dei laboratori, gestione degli elementi di gioco e reportistica. In Figura 4.3 sono mostrati i principali casi d'uso che riguardano l'amministatore.

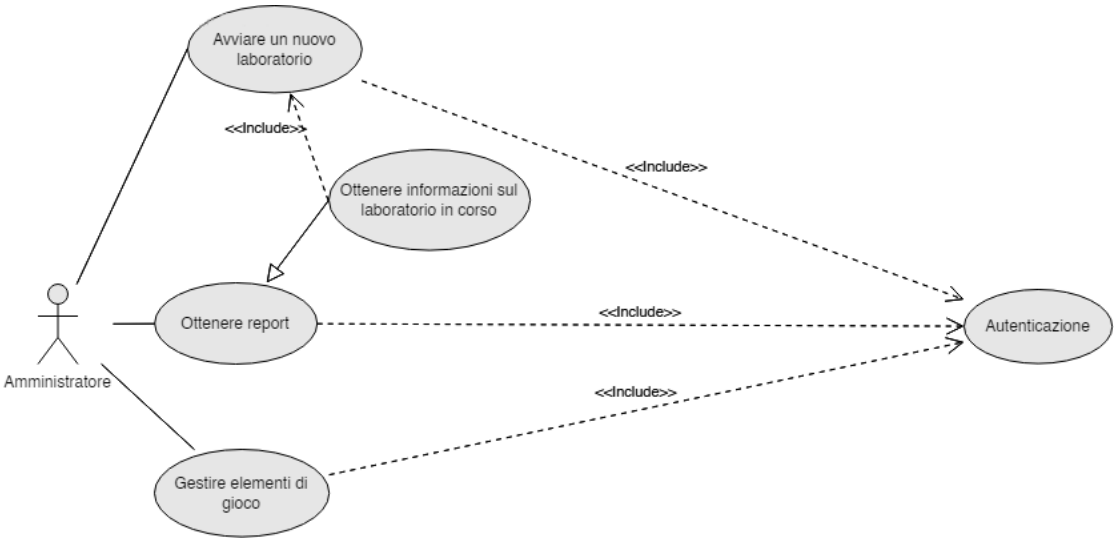

Figura 4.3: Rappresentazione per mezzo di uno Use Case Diagram UML dei principali casi d'uso che coinvolgono l'amministratore.

A titolo esemplificativo viene mostrato in Figura 4.5 un diagramma raffigurante il caso d'uso di ottenimento di informazioni sugli gli obiettivi del giocatore, che a sua volta coinvolge un caso d'uso proprio dell'amministratore.

4.3 I prototipi

Lo studio dei requisiti e la loro strutturazione in casi d'uso hanno permesso di ideare l'organizzazione delle interfacce dei due frontend web e del flusso di esecuzione dei vari task. Tali elementi sono stati concretizzati sotto forma di prototipi di carta (low-fidelity) per essere "eseguiti a mano" in modo da individuare eventuali criticità. Il risultato è stato infine trasformato in wireframe (high-fidelity) e sarà descritto nelle sezioni a seguire. [18]

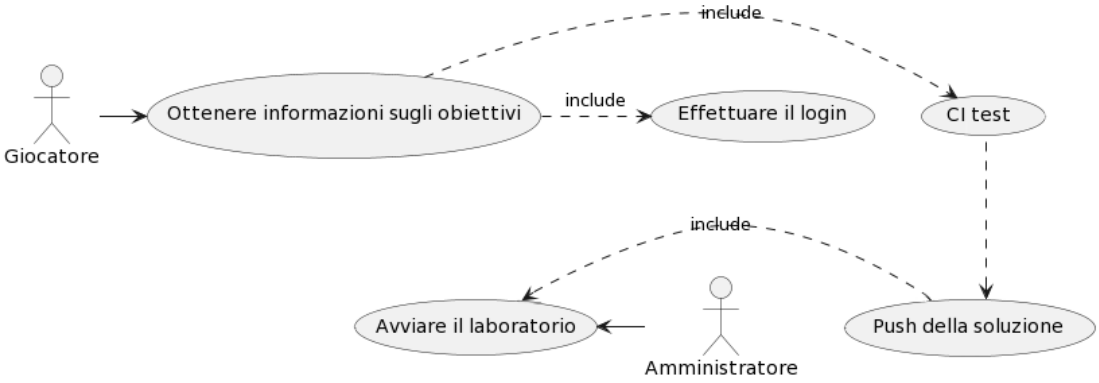

Figura 4.4: Rappresentazione per mezzo di uno Use Case Diagram UML del caso d'uso di ottenimento delle informazioni sui propri obiettivi.

4.3.1 L'interfaccia del frontend per i giocatori

Seguendo la suddivisione in sezioni dei requisiti riguardanti l'utente di tipo giocatore, l'applicazione frontend presenta un menu di navigazione principale contenente le voci "Leaderboard", "Labs", "Shop", "About" e "Profile". Il menu è situato su una barra di navigazione superiore, punto di accesso alle varie sezioni comune a tutte le interfacce dell'applicazione.

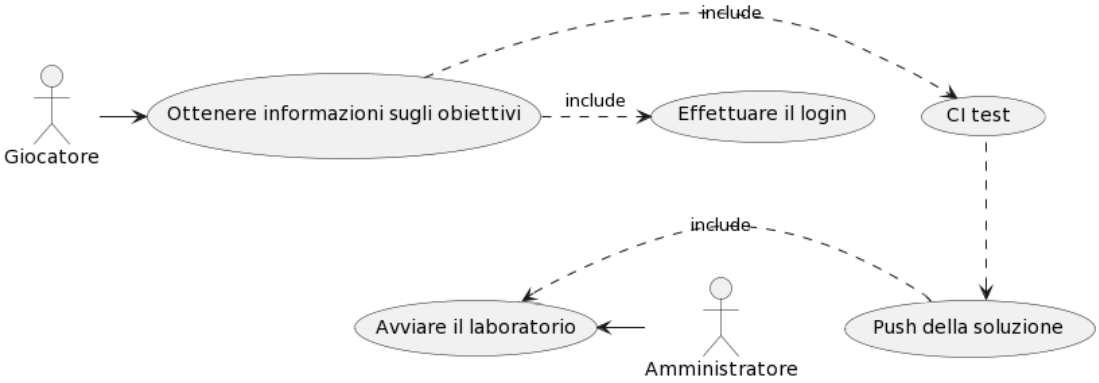

Figura 4.5: Rappresentazione per mezzo di uno Use Case Diagram UML del caso d'uso di ottenimento delle informazioni sui propri obiettivi.

Il primo dei menu – "**Leaderboard**" – mostra le prime 10 posizioni della classifica globale (calcolata come specificato in fase di design). Ad ogni giocatore in classifica viene associata la posizione, l'immagine del profilo, il nickname e i punti guadagnati in quel laboratorio. L'analisi di tale prototipo ha permesso di scovare una mancanza nell'interfaccia: un giocatore non rientrante nelle prime 10

posizioni non avrebbe modo di visionare la propria collocazione nella classifica globale. La falla è stata colmata in fase di implementazione inserendo nella sezione "Leaderboard" un'indicazione della propria posizione e di quelle immediatamente successiva e precedente (oltre alla classifica globale citata in precedenza).

La sezione "**Labs**" contiene informazioni generali sui laboratori passati e su quello in corso (nel caso in cui ce ne fosse uno). Tali laboratori sono ordinati e navigabili attraverso un menu laterale fisso, che li distingue fra "completati" e "in corso" (nel caso in cui fosse presente, sarebbe sempre in cima alla lista) attraverso apposite icone. Il contenuto della parte centrale della pagina varia in base al laboratorio selezionato. Nel caso del laboratorio in corso, vengono mostrati nome del laboratorio, data di scadenza e traccia; in particolare, seguendo quanto prestabilito in fase di design, la data di scadenza viene inizialmente mostrata in forma testuale e sarà convertita in un conto alla rovescia a partire da 24 ore prima della scadenza del laboratorio. Passando a un laboratorio terminato (Figura 4.6), saranno consultabili – oltre al nome del laboratorio, alla data in cui scadde e alla traccia – anche la classifica relativa a quel laboratorio (contenente le prime dieci posizioni) e i risultati personali (posizione e punteggio) dell'utente che sta visitando la pagina.

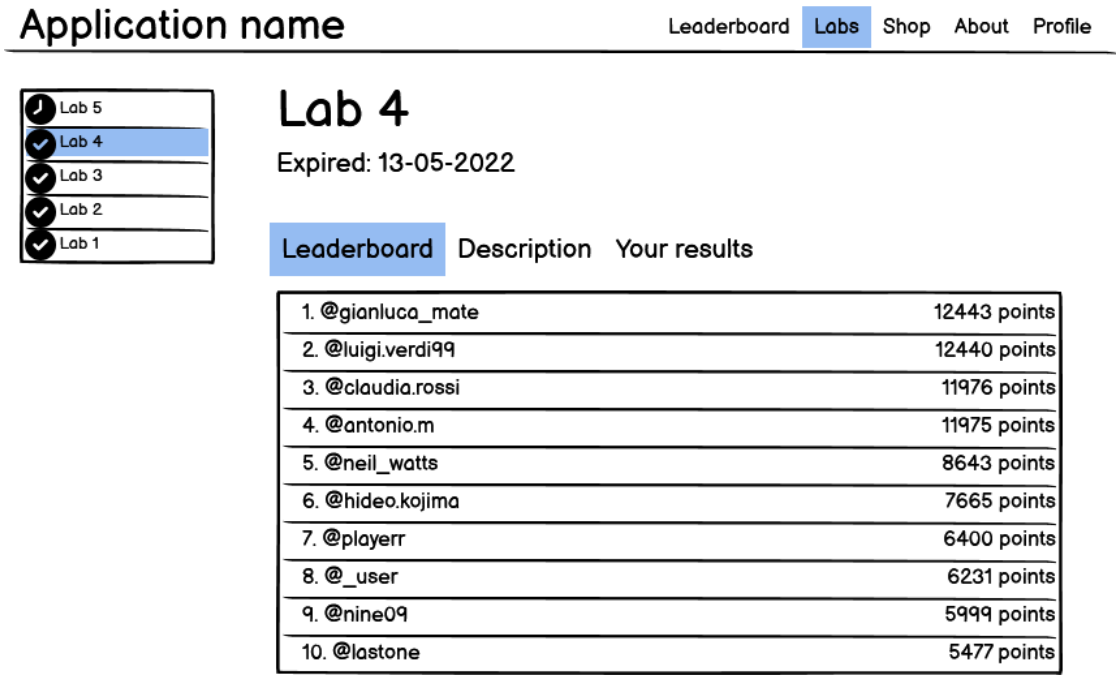

Figura 4.6: Wireframe raffigurante l'interfaccia che mostra la classifica parziale di un laboratorio passato.

Attraverso la sezione "**Shop**" (Figura 4.7), il giocatore può sfruttare le monete

acquisite per comprare nuovi avatar da aggiungere alla propria collezione o da usare come immagine del profilo. Gli avatar disponibili all'acquisto vengono mostrati in una struttura a griglia che si rifà idealmente a layout comprovati di store digitali. Ogni elemento della griglia mostra l'immagine dell'avatar, il nome e il prezzo; la descrizione dell'avatar (informazione reputata secondaria rispetto alle precedenti) può essere visualizzata attraverso il bottone con l'icona di informazioni, accanto al quale è presente il bottone di aggiunta al carrello. Per mezzo di un bottone situato nella parte inferiore della pagina (in stile "floating action button") si può accedere al riepilogo dell'ordine: una vista contenente gli avatar selezionati (con le rispettive informazioni associate e con la possibilità di rimuoverli dal carrello), il totale parziale e il bottone per il completamento dell'acquisto. Quest'ultimo mostrerà un avviso se cliccato quando il carrello è vuoto, altrimenti sarà preceduto – seguendo le linee guida dell'interazione uomo-macchina – da una richiesta di conferma in modo da permettere di tornare indietro a utenti che cliccano per errore. L'utente viene notificato del completamento dell'ordine tramite un avviso che compare temporaneamente al centro della finestra.

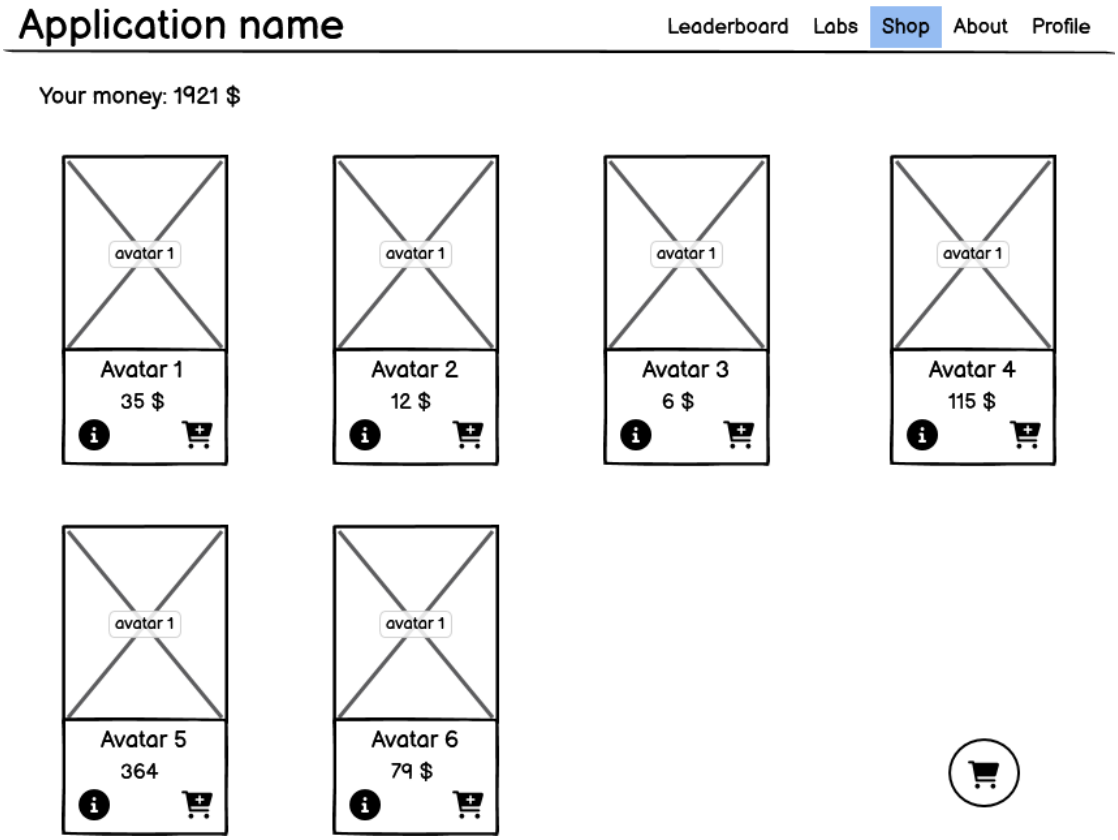

Figura 4.7: Wireframe raffigurante l'interfaccia della sezione "Shop".

In "**About**" è possibile consultare le regole di gioco e una guida circa il funzionamento dell'applicazione.

Nella sezione "**Profile**" (Figura 4.8) si è deciso di raggruppare tutte le informazioni che riguardano direttamente l'utente. Tale sezione è suddivisa in due fasce: una superiore e statica e una inferiore e dinamica. Nell'area superiore dell'interfaccia vengono mostrate le informazioni personali (nome, cognome e nickname), la foto del profilo (scelta a discrezione dell'utente fra gli avatar in proprio possesso), la quantità di monete accumulate e il numero di obiettivi raggiunti. Il contenuto della fascia inferiore è navigabile attraverso un menu laterale, tramite il quale l'utente può prendere visione dei propri risultati (completamento, posizione in classifica e punti ottenuti) riguardanti i laboratori passati, degli avatar in suo possesso e degli obiettivi. Questi ultimi possono risultare completati o da completare: nel primo caso sarà visibile l'immagine del badge associato all'obiettivo, mentre nel secondo caso l'immagine sarà oscurata e sarà presente (in alcuni casi) una progress bar associata alla percentuale di completamento. Dalla sezione "Profilo" l'utente potrà scegliere quale fra gli avatar in proprio possesso usare come immagine del profilo.

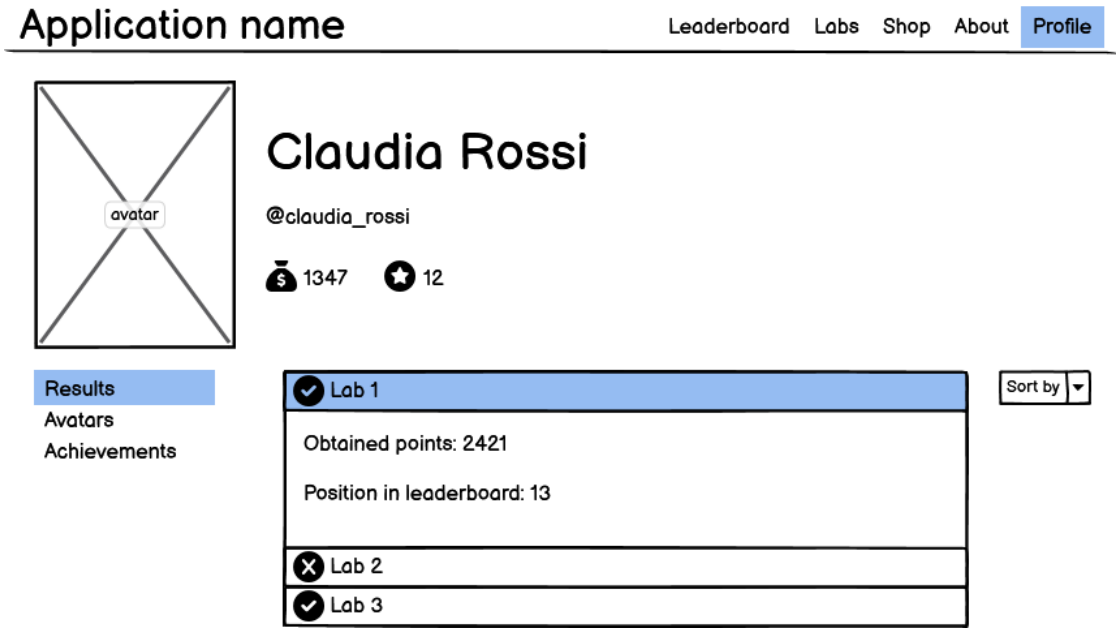

Figura 4.8: Wireframe raffigurante l'interfaccia della sezione "Profile" contenente lo storico dei risultati dell'utente.

4.3.2 L'interfaccia del frontend per gli amministratori

Similmente a quanto fatto per il corrispettivo per giocatori, anche l'interfaccia per amministratori presenta una barra superiore sulla quale sono situati i collegamenti alle principali sezioni dell'applicazione: "Leaderboard", "Labs", "Avatars" e "Reports".

Il menu "Leaderboard" permette di visualizzare tutte le posizioni della classifica globale. Ad ogni giocatore viene associata la posizione, l'avatar, il nome utente, il nome completo e i punti accumulati.

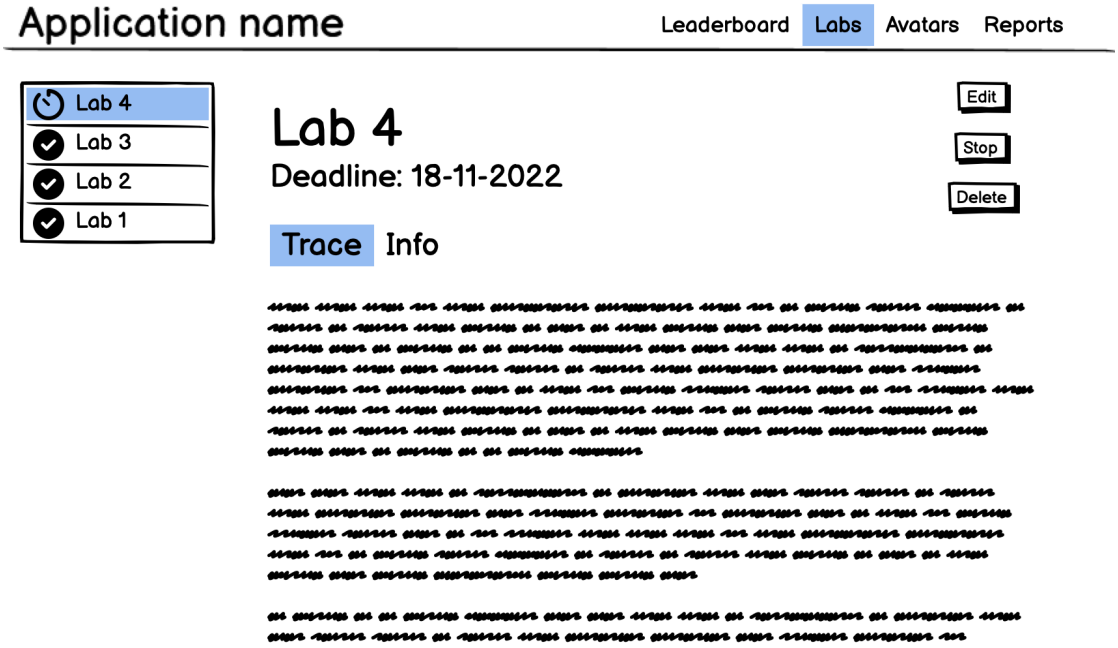

Figura 4.9: Wireframe raffigurante l'interfaccia (admin) della sezione "Labs" se selezionato un laboratorio in corso.

Nella sezione "Labs" è possibile consultare i laboratori passati ed eventualmente quello in corso (a distinguerli sarà l'icona associata al nome nel menu di navigazione laterale, come mostrato in Figura 4.9). La parte centrale della pagina mostra informazioni riguardanti lo specifico laboratorio: se quest'ultimo è terminato, verranno mostrati, unitamente al nome del laboratorio e alla data di scadenza, la traccia, le informazioni (id del laboratorio, numero di partecipanti, numero massimo di test e link alla repository contenente la soluzione) e la classifica totale; se è ancora in corso non si potrà consultare la classifica. Nel caso in cui non fosse presente un laboratorio in corso, un avviso ricorderà all'amministratore di poter avviarne uno cliccando sul bottone (in stile "floating action button") all'estremità

inferiore della pagina, il quale permette di aprire una finestra contenente campi (validati prima della conferma) per immettere nome del laboratorio, traccia, data di scadenza, numero massimo di test che gli utenti possono scrivere e link alla repository contenente la soluzione del laboratorio. Ogni laboratorio può essere modificato o eliminato per mezzo di bottoni presenti nella sezione centrale della pagina. Ogni bottone aprirà una finestra di conferma ed eventualmente con campi il cui contenuto verrà validato prima della sottomissione. Ogni finestra, inoltre, è predisposta a ricevere errori dal backend, che verranno segnalati tramite un avviso a schermo. L'eventuale laboratorio in corso può essere interrotto tramite un bottone in maniera simile a quanto descritto pocanzi.

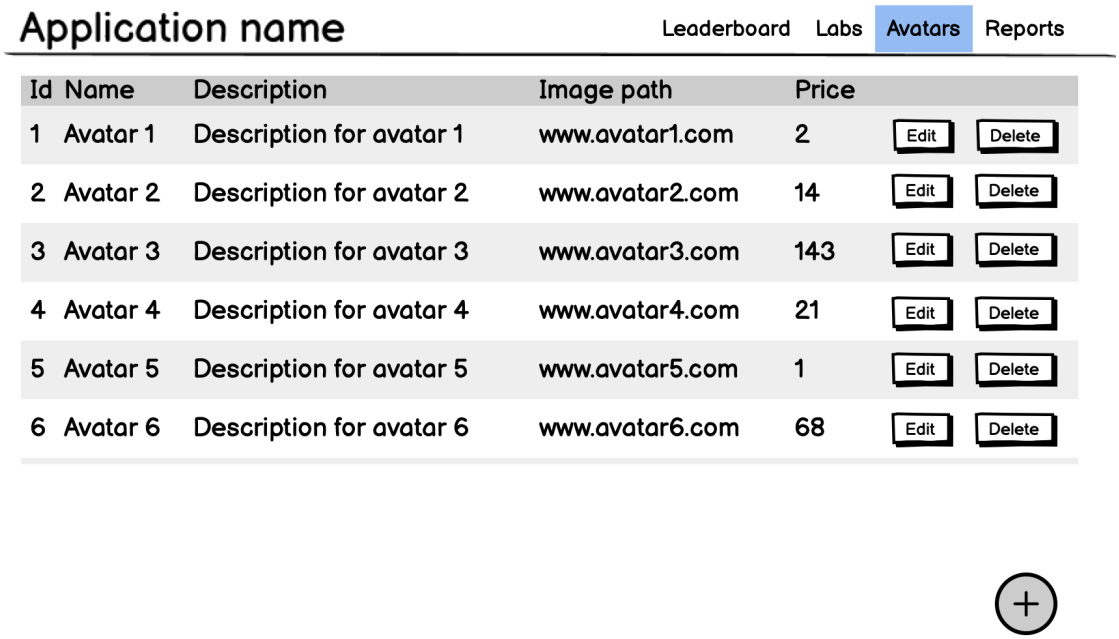

Figura 4.10: Wireframe raffigurante l'interfaccia (admin) della sezione "Avatars".

Tramite la sezione "Avatars" (Figura 4.10) è possibile consultare l'elenco degli avatar che gli utenti possono visualizzare e acquistare. Ad ogni avatar è associato un id, il nome, la descrizione, il link all'immagine e il prezzo. L'amministratore può aggiungere un nuovo avatar tramite il bottone "+" situato nella parte inferiore della pagina, alla pressione del quale si aprirà una finestra con i campi per l'inserimento delle informazioni (tutti validati prima della conferma). È inoltre possibile modificare le proprietà associate a un determinato avatar o eliminarlo completamente: nel primo caso si aprirà una finestra con i campi da modificare, nel secondo una finestra di conferma per evitare che vengano effettuate eliminazioni indesiderate.

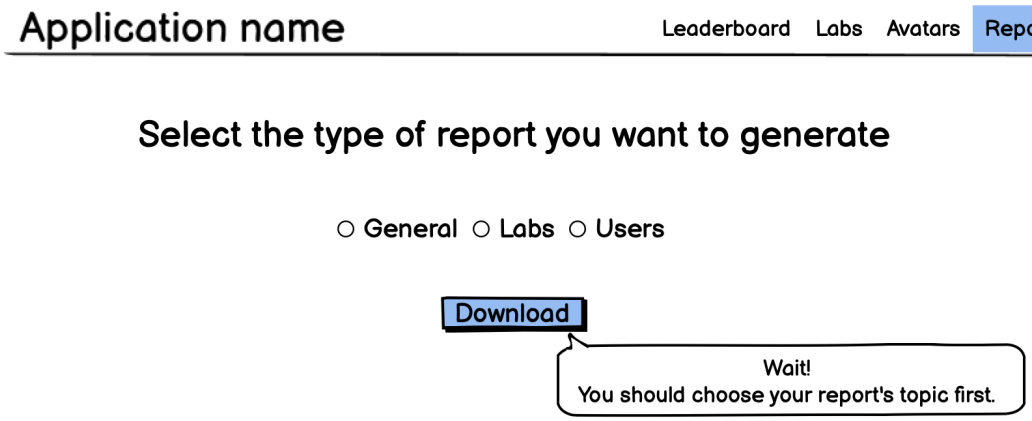

Figura 4.11: Wireframe raffigurante l'interfaccia (admin) della sezione "Reports" al clic del bottone senza aver selezionato alcun tipo di report.

L'ultima sezione, "Reports", permette all'amministratore di scaricare report sotto forma di file Excel. L'interfaccia propone – per mezzo di una serie di "radio buttons" – una scelta sul tipo di report da scaricare:

- "General", report contenente informazioni sullo stato generale del sistema (numero di laboratori totali, numero di utenti totali, media di partecipanti per laboratorio, media punti per laboratorio);
- "Labs", report contenente informazioni sui singoli laboratori (per ogni laboratorio: id, numero di partecipanti, percentuale di partecipanti sul totale di utenti, media del punteggio per partecipante);
- "Users", report contenente informazioni sui singoli utenti (per ogni utente: id, nome utente, nome completo, numero di laboratori a cui ha partecipato, percentuale di laboratori a cui ha partecipato, media dei punteggi per laboratorio e miglior posizionamento in classifica).

È possibile scaricare il report scelto tramite il bottone "Download", che mostrerà un avviso se non è stato selezionato alcun report (Figura 4.11).

In seguito all'analisi del prototipo si è ritenuto utile aggiungere un ulteriore tipologia di report da fornire all'amministratore, le cui righe contengono informazioni

sui risultati dei singoli laboratori (id utente, id laboratorio, punti, posizione, URL del repository, percentuale di copertura, numero totale di test falliti su soluzioni avversarie, numero totale di test avversari falliti sulla soluzione dell'utente).

4.4 L'applicazione

4.4.1 Frontend per i giocatori

L'applicazione frontend destinata all'interazione con i giocatori presenta i seguenti elementi principali:

- il **file index.js**, punto di avvio dell'applicazione, si occupa di renderizzare il componente App;
- il **file App.js**, definisce la struttura generale dell'applicazione: una Navbar nella parte superiore e una parte inferiore in cui si alterneranno componenti in base alla route in cui si trova l'utente. Tale navigazione avviene per mezzo del componente Router;
- il **file App.css**, contenente il codice css (in parte predefinito e in parte scritto ad hoc) per la personalizzazione degli elementi dell'interfaccia;
- il **file API.js**, implementa le funzioni per effettuare chiamate HTTP al back-end, esposte per l'uso da parte dei vari componenti per ottenere dati da mostrare a schermo o per inviare al server dati sottomessi dall'utente;
- il **package models**, al cui interno sono definiti i modelli dei principali tipi di dati gestiti dall'applicazione;
- il **package components**, che contiene tutti i componenti dell'interfaccia utente.

All'interno del package *components* sono presenti innanzitutto i componenti MyNavbar e HomePage: il primo definisce la barra di navigazione superiore, il secondo la pagina di benvenuto. I restanti componenti sono organizzati in package a seconda del loro contesto di utilizzo.

La cartella *common* contiene i componenti utilizzati in più parti dell'applicazione. Al suo interno si trova Leaderboard.js, che definisce il componente che organizza in una tabella a mo' di classifica (per mezzo del componente Table) una lista di risultati ricevuti in input, creando dinamicamente le righe in base ad essi. Un primo utilizzo della Leaderboard si ha nel componente LeaderboardSection del package *leaderboard*, utilizzato per mostrare la classifica generale dei giocatori (Figura 4.13).

Nel package *labs* sono contenuti i componenti che vengono visualizzati nella sezione "Labs" (Figura 4.14). La struttura di quest'ultima è definita nel file Labs.js,

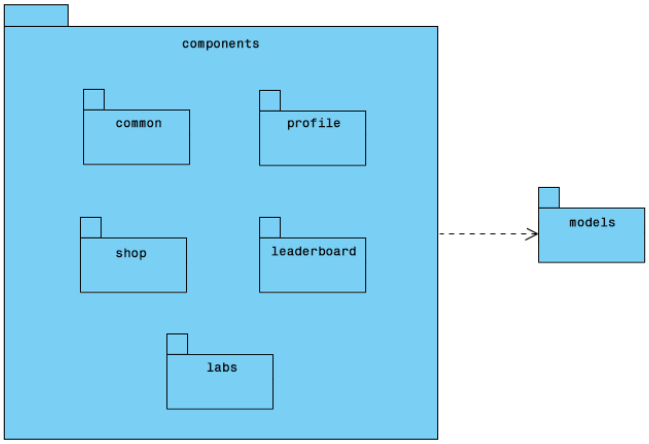

Figura 4.12: Package diagram UML rappresentante la suddivisione interna del progetto del frontend per giocatori.

che suddivide la pagina nei componenti LabList e LabMain seguendo quanto considerato in fase di design. LabList non è altro che un ListGroup i cui elementi vengono creati dinamicamente in base alla lista di laboratori ricevuta dal back-end e contiene al suo interno la logica per la navigazione dei LabMain. Il menu di navigazione presente all'interno di quest'ultimo è costituito da un Nav che alterna a schermo i componenti che definiscono le sezioni. In particolare, in "Leaderboard" viene fatto nuovamente uso di Leaderboard.js. All'interno di LabMain viene inoltre sfruttata la libreria "Day.js" per manipolare entità rappresentanti date: le sue funzionalità permettono di sostituire la stringa indicante la data di scadenza di un laboratorio in corso con una progress bar (componente ProgressBar di react-bootstrap) e un conto alla rovescia (componente react-countdown) da 24 ore prima della scadenza del laboratorio, come stabilito in fase di design. Il colore della progress bar varierà dal giallo al rosso all'avvicinarsi della scadenza.

Il package *profile* contiene gli elementi che compongono la sezione "Profile" (Figura 4.15). Anche qui si ha un menu di navigazione laterale costituito da un ListGroup, che gestisce la navigazione fra ProfileSectionResults, ProfileSectionAvatars e ProfileSectionAchievements. Il primo rappresenta la sezione contenente lo storico dei risultati dell'utente, costituiti da una lista di Accordion espandibili per visualizzare le informazioni; il secondo non è altro che una griglia di Card contenenti gli avatar in possesso dell'utente; il terzo contiene – come in LabMain – un Nav, usato in questo caso per filtrare la lista di obiettivi.

Infine, nel package *shop* sono presenti i componenti presenti nella sezione Shop. L'interfaccia è costituita da una griglia di Card, ma sono stati sfruttati anche un

#	Player	Points
1	 Mattia	14000
2	 Clodia	13965
3	 ant	13930
4	 Neil	7400
5	 jack_99	2314
6	 Wander	1985

Figura 4.13: Sezione di leaderboard globale con posizioni, avatar, nickname e punteggi dei giocatori.

bottone in stile "floating action button" (dal package react-floating-action-button), un Offcanvas per mostrare gli elementi presenti nel carrello e vari Modal per aprire ulteriori finestre in caso di errori o conferma dell'ordine. Il bottone "Buy" è associato a un Tooltip che mostra un avviso in caso di clic quando il carrello è vuoto. Inoltre, viene utilizzato un Toast legato a un Overlay in modo da dare all'utente un messaggio di conferma dell'ordine andato a buon fine.

In tutte le sezioni in cui ne viene fatto uso, le immagini raffiguranti gli avatar sono state create per mezzo di "avataaars generator" [1], sito web che mette a disposizione un editor per mezzo del quale è possibile generare (anche sotto forma di link) avatar personalizzabili nelle loro componenti (colore dei capelli, indumenti, accessori, ecc.).

4.4.2 Frontend per amministratori

L'applicazione frontend per l'amministratore è stata strutturata in maniera simile al corrispettivo per giocatori. Le differenze sostanziali possono essere trovate nei

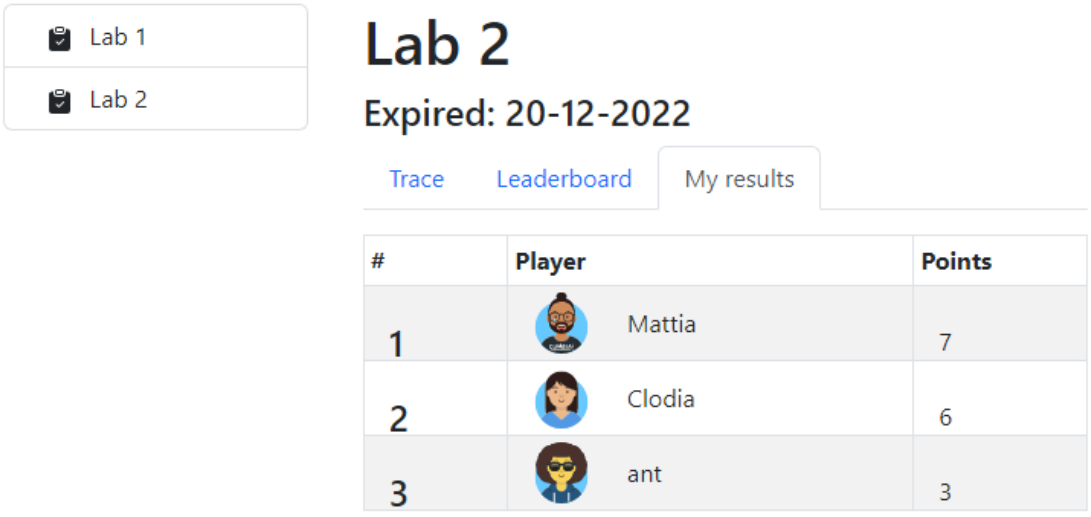

Figura 4.14: Esempio della sezione "Lab", in cui vengono mostrate la posizione di Clodia e quelle immediatamente precedenti e successive.

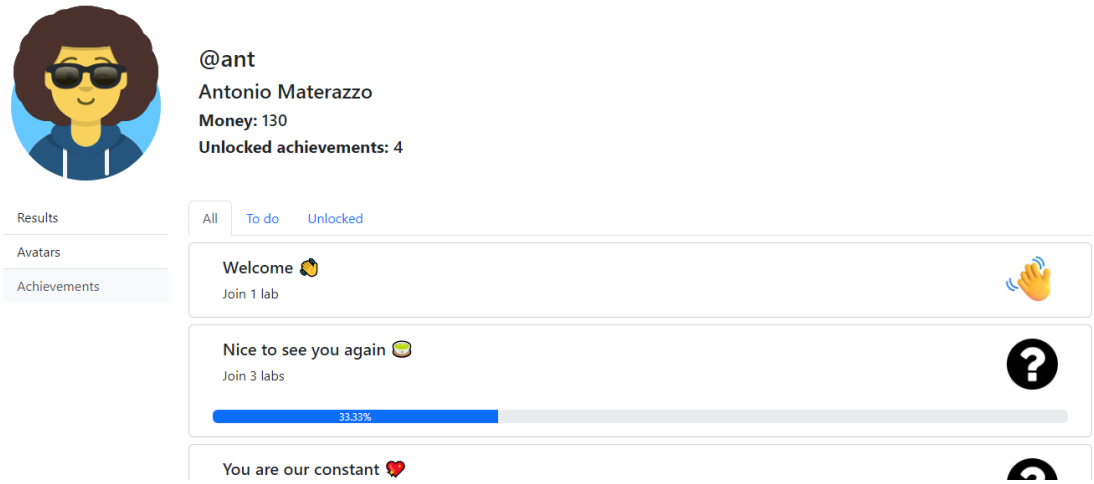

Figura 4.15: Esempio della sezione "Profile", in cui vengono mostrati gli obiettivi con le rispettive percentuali di completamento.

package "models" e "components". Il primo definisce i modelli di entità che verranno gestiti da questa specifica applicazione, mentre il secondo contiene i componenti che si susseguiranno nel corso delle interazioni con l'amministratore. All'interno di quest'ultimo, i package "common" e "leaderboard" si comportano in maniera similare ai loro corrispettivi descritti nel sottoparagrafo precedente.

Anche il package "labs" risulta simile al suo corrispettivo nell'applicazione per

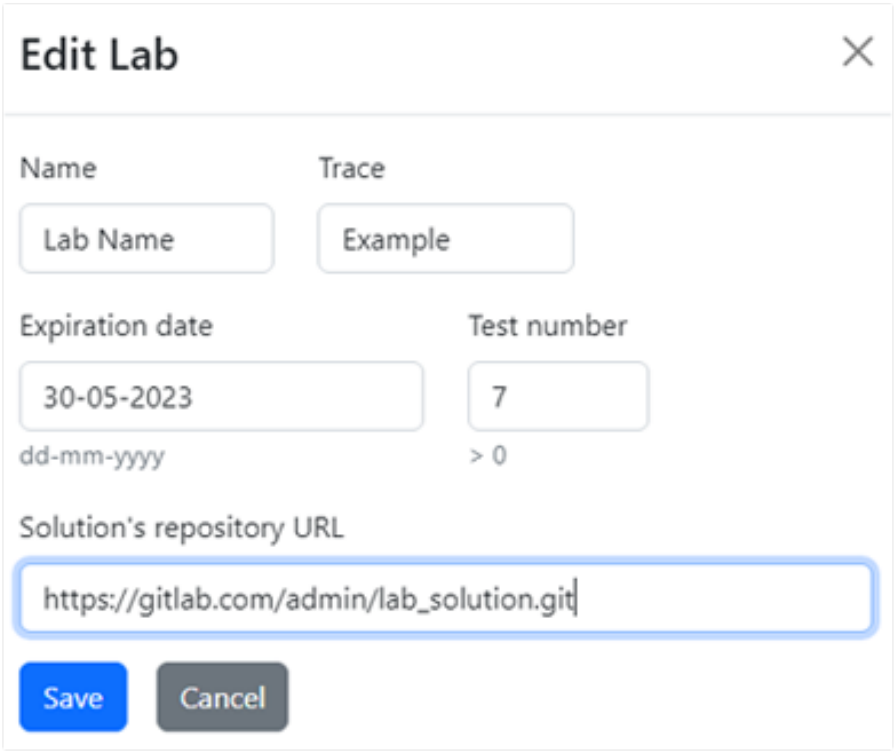

Figura 4.16: Dettaglio della finestra di modifica di un laboratorio.

utenti, con l'eccezione che è stata aggiunta la logica per gestire l'aggiunta di un nuovo laboratorio o la modifica, interruzione o eliminazione di un laboratorio in corso. Al momento dell'aggiunta o modifica di un laboratorio viene aperta una finestra (Figura 4.16) contenente campi di testo (vuoti o pre-riempiti) per l'immissione delle informazioni riguardanti il laboratorio. La verifica del loro contenuto avviene ad ogni cambiamento dello stesso per mezzo di apposite funzioni di controllo: ogni campo di testo non può essere vuoto, la data di scadenza deve essere successiva al giorno corrente e il numero massimo di test che possono essere scritti dagli utenti non può essere minore o uguale a zero.

La sezione "Avatars" è gestita nel file Avatars.js, il quale costruisce una tabella (componente Table di react-bootstrap) sulla base dei dati ottenuti tramite l'apposita funzione di API.js. Anche qui viene fatto uso del componente Modal: nei casi di aggiunta e modifica di un avatar viene riutilizzato lo stesso, ma con contenuto

riempito diversamente. Per quanto riguarda i controlli effettuati, i campi di testo non possono essere vuoti e il prezzo non può essere minore di zero.

Infine, il file Reports.js contiene il codice della sezione "Reports". Al suo interno si fa uso di una serie di "radio buttons" (componente Form di react-bootstrap) mutuamente esclusivi per la scelta del tipo di report e della combinazione dei componenti OverlayTrigger e Popover per mostrare un avviso nel momento in cui si fa clic sul bottone senza aver selezionato un tipo di report. Una volta ottenuti i dati tramite le funzioni di API.js, viene generato e scaricato un file con estensione ".xlsx" grazie all'utilizzo del modulo "xlsx" disponibile per React.

4.4.3 Backend

Come chiarito precedentemente, il backend funge da punto di comunicazione fra i vari componenti, fornendo API per l'accesso ai dati (organizzati come mostrato in Figura 4.17) e per la loro modifica e implementando la logica riguardante i processi di gioco. Il server è così strutturato:

- il file **index.js**, punto di bootstrap dell'applicazione;
- il file **database.db**, database contenente i dati del sistema;
- il **package routes**, al cui interno sono presenti i file che espongono le route del sistema (per mezzo dell'elemento "Router" fornito dal modulo "express"), collegandole ai rispettivi controller;
- il **package middlewares**, funzioni per la gestione di alcuni aspetti delle richieste ricevute non riguardanti il contenuto della richiesta stessa (ad esempio l'autenticazione e autorizzazione dell'utente richiedente);
- il **package controllers**, punto di raccolta dei controller, componenti che gestiscono tutte le operazioni legate al protocollo HTTP, rimandando ai service il soddisfacimento del contenuto della richiesta;
- il **package services**, contenente la logica di business dell'applicazione per soddisfare le richieste in arrivo e per implementare i processi di gioco;
- il **package daos**, i cui file implementano le query per l'accesso al database;
- il **package models**, per la definizione dei modelli di alcuni tipi di dati ricorrenti;
- il **package config**, al cui interno sono presenti file di configurazione (ad esempio quello riguardante il database);
- il **package utils**, contenente funzioni di utilità generale.

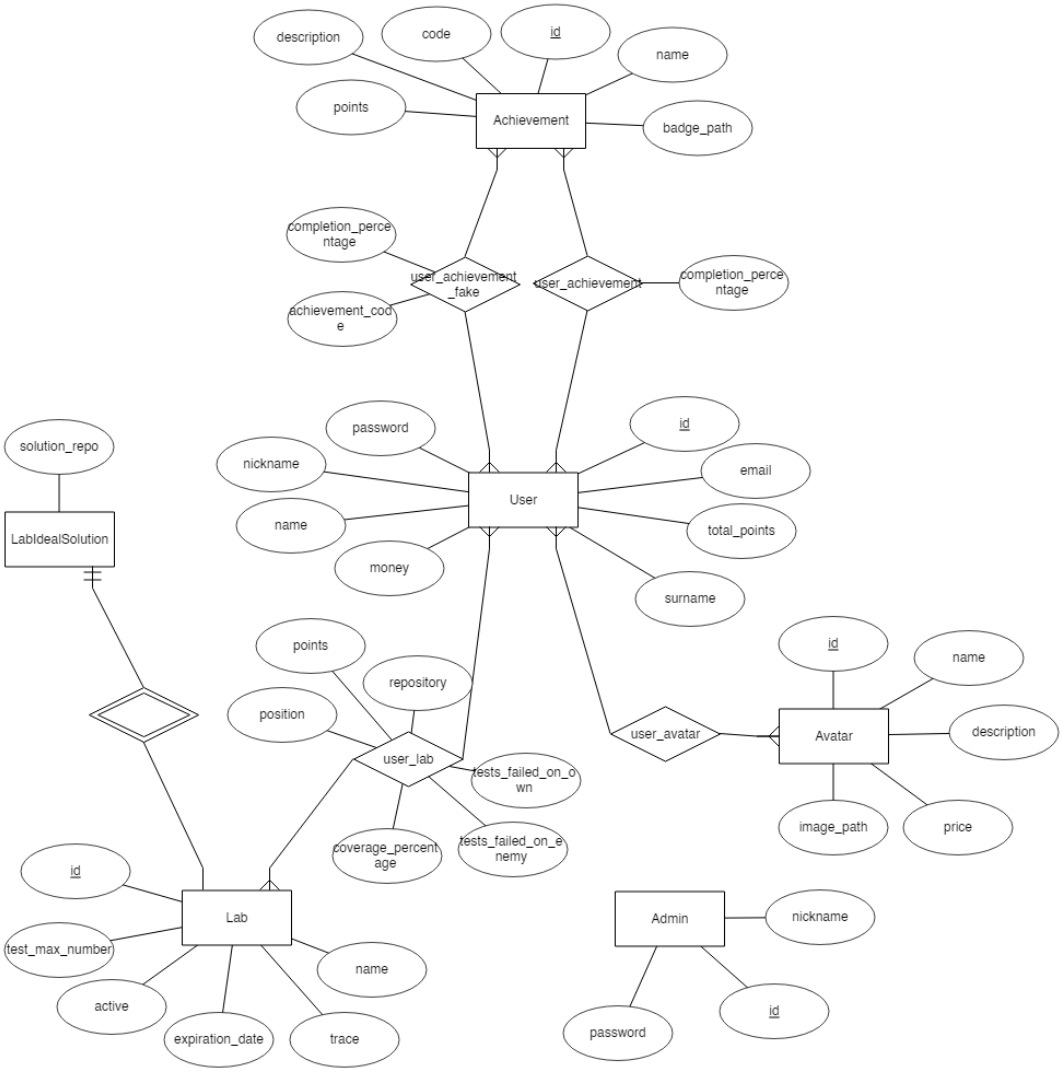

Figura 4.17: Diagramma ER che mostra l'organizzazione del database.

Di particolare interesse risultano i file contenuti nel package *services*. Alcuni di questi (ad esempio userService.js) non sono altro che ponti fra i package *controllers* e *dao*: demandano alle query definite nei file del secondo l'effettivo soddisfacimento delle richieste ricevute e definiscono le eccezioni da lanciare e da far intercettare ai controller per differenziare i casi di errore. Fra questi, reportService.js contiene la logica più complessa, dal momento che calcola le metriche richieste nei report per l'amministratore.

Altri invece (come shellService.js o fileService.js) forniscono un livello di astrazione su alcune funzionalità richieste da altri componenti riguardanti l'utilizzo di specifici elementi o librerie. Infine, il file gameService.js implementa gli algoritmi che regolano i processi di gioco, in particolar modo quelli riguardanti la validazione delle soluzioni e il calcolo dei punteggi alla fine di ogni laboratorio. Segue una descrizione più approfondita di alcuni service significativi.

Il file shellService.js espone funzioni che eseguono comandi in una shell. Il tutto avviene per mezzo del modulo "child_process", che permette di avviare sottoprocessi: il metodo "execSync", in particolare, avvia una shell e permette di eseguire comandi in essa, bloccando nel mentre l'event loop di Node.js. Le funzioni esposte dal file permettono di compilare e testare un progetto Maven all'interno di un container Docker e di clonare repository git a partire da un link.

Il file fileService.js si occupa di operazioni che riguardano strettamente la gestione di file. Molte di esse si sarebbero potute implementare per mezzo di comandi shell in shellService.js, ma si è scelto di implementarle in questo contesto sfruttando l'astrazione fornita dai moduli "fs" e "fs-extra" (il secondo è un'estensione del primo). Entrambi espongono funzioni per la gestione del file system, grazie alle quali il componente in esame permette la copia di file, la creazione, pulizia ed eliminazione di cartelle e operazioni simili. Sono inoltre presenti funzioni riguardanti il parsing di file XML, che fanno uso di un ulteriore modulo: "fast-xml-parser".

Infine vi è il file gameService.js, che espone la funzione finalProcess per eseguire tutte le operazioni stabilite in fase di design una volta superato il termine di un laboratorio. La funzione, dopo aver clonato la soluzione ideale e le soluzioni degli utenti, procede a filtrare queste ultime, assegnando 0 punti ed eliminando le soluzioni:

- con errori di sintassi (che presentano quindi errori di compilazione);
- i cui test falliscono se lanciati sulla soluzione ideale (per cui sono plausibilmente presenti errori nei test stessi);
- che contengono un numero di test superiore a quello concesso per il laboratorio.

Alle soluzioni che superano i controlli viene assegnato 1 punto di partenza, come anticipato nei capitoli precedenti, e vengono passate come parametro alla funzione beginWar, che si occupa della loro valutazione. Innanzitutto, per ogni soluzione rimasta, viene eseguita prima la funzione testSelf per l'aggiornamento delle percentuali di completamento degli obiettivi. In particolare, viene fatto uso della libreria JaCoCo per ottenere dati riguardanti la percentuale di copertura del proprio codice raggiunta dai propri test. In seguito, per ogni soluzione di utenti avversari, si calcola il punteggio relativo all'esito dei propri test eseguiti su tali soluzioni tramite la funzione testVS. Si conclude il processo con l'assegnazione delle posizioni nel

singolo laboratorio e con l'aggiornamento dei punteggi globali e delle percentuali di completamento degli obiettivi restanti, come stabilito in fase di design.

```
const finalProcess = async () => {
    //recupero il laboratorio in corso
    const lab = await getCurrentLab();
    const labId = lab.id;
    //chiudo il laboratorio in corso
    await closeLab(lab);
    //recupero le informazioni degli utenti iscritti al
   laboratorio
    const userLabs = await labDao.getUserLabListByLabId(labId);
    //clono la soluzione ideale nel percorso prestabilito
    const linkToIdealSolution = await labDao.
   getLinkToIdealSolution(labId);
    fileService.clearDirectory(rootIdealsolution)
    await shellService.cloneIdealSolution(linkToIdealSolution,
   rootIdealsolution);
    //clono i repository degli utenti iscritti e ottengo le
   soluzioni
    const solutions = await getSolutionsFromUserLabList(userLabs);
    //filtro le soluzioni in base ai test riguardanti compilazione
   , soluzione ideale e numero di test
    const filteredSolutions = await filterSolutions(solutions);
    //inizio il processo di test e assegno i punti
    await beginWar(filteredSolutions);
    //assegno le posizioni e aggiorno la classifica globale
    await assignPositionsAndUpdateGlobalLeaderboard(labId)
    //aggiorno gli achievements
    await updateAchievements(labId)
}
```

Listing 4.2: Funzione finalProcess

Le operazioni di test, clone, accesso ai file e parsing degli stessi vengono effettuate grazie all'appoggio dei file shellService.js e fileService.js descritti precedentemente. Il file gameService.js definisce inoltre una funzione che ogni mezzanotte si accerta di rendere inattivi eventuali laboratori che abbiano raggiunto la data di scadenza, avviando il processo precedentemente descritto. Tale comportamento si ottiene per mezzo del modulo node-schedule, che permette di programmare l'esecuzione di task a piacimento.

4.4.4 Registrazione e login

Registrazione e login vengono descritti in quest'ultimo sottoparagrafo in quanto implementati in maniera quanto più logicamente separata possibile dal resto dei componenti, sia per i frontend che per il backend. Si anticipa che la mancata

implementazione di alcuni importanti aspetti di sicurezza sarà oggetto di discussione in merito agli sviluppi futuri del sistema, in quanto fuori dagli scopi principali del presente lavoro di tesi.

Per quanto riguarda il frontend destinato ai giocatori, solo alcune sezioni (homepage, about, login e registrazione) sono accessibili da utenti non loggati, mentre tutte le altre sono protette in modo da reindirizzare utenti non ancora autenticati alla pagina di login . Quest'ultima consiste principalmente in due campi, attraverso i quali l'utente può inserire username e password scelti in fase di registrazione per autenticarsi. È presente inoltre un link alla pagina di registrazione, la cui struttura è simile alla precedente, ad eccezione della presenza di ulteriori campi per l'inserimento di nome, cognome e indirizzo email. La logica e implementazione delle due pagine – i cui campi sono tutti obbligatori e alcuni dei quali sono validati (la password deve essere superiore ai 6 caratteri, l'email deve rispettare il formato definito da un'apposita regex) – descritte poc'anzi sono definite dai file LoginPage e RegisterPage del package "login". L'effettiva registrazione o login avviene per mezzo di apposite API; l'esito affermativo della seconda permette di salvare uno stato (tramite l'hook "useState" di React) utilizzato nel componente App per determinare quali route rendere accessibili e propagato al componente Navbar in modo da determinare quali voci del menu di navigazione mostrare o nascondere. La protezione garantita dal login non coinvolge unicamente i componenti grafici: le credenziali salvate vengono associate ad ogni richiesta effettuata dal componente API, in modo che il backend possa riceverle e verificarle per proteggere le route esposte.

Per quanto concerne il frontend per gli amministratori, è stata ipotizzata una situazione diversa. Non è presente alcun componente di registrazione, in quanto l'amministratore dovrà autenticarsi (tramite una pagina di login identica a quella descritta precedentemente) tramite un nome utente ("admin") e una password inseriti manualmente all'interno del database.

Conseguentemente a quanto illustrato finora, il lavoro riguardante il backend si è svolto su due fronti: protezione delle route e implementazione di API per il supporto di registrazione, login e logout. Per la strutturazione di queste ultime si è seguita la stratificazione descritta in precedenza (route-controller-service-dao), mentre per il primo fronte si è fatto largo uso di Passport, un middleware disponibile per Node.js che permette di scegliere una o più strategie di autenticazione: in questo caso è stata scelta la strategia "passport-local", basata sull'autenticazione per mezzo di nome utente e password (salvati all'interno del database in seguito a registrazione). Passport è stato usato inoltre per serializzare e deserializzare alcune informazioni dell'utente loggato nella sessione corrente (come l'id, accessibile così dai controller), precedentemente configurata attraverso il middleware express-session. La logica appena descritta è implementata dai metodi use, serializeUser e deserializeUser nel file login del package middlewares; lo stesso file esporta due middleware definiti ad

hoc, isLoggedIn e isAdmin: il primo verifica che la richiesta appena ricevuta sia stata effettuata da un utente autenticato, la seconda verifica che lo stesso utente sia un admin (tramite le informazioni salvate nella sessione). Tali middleware vengono sfruttati in index.js per proteggere le route in base ai diritti d'accesso che ognuna di queste richiede: per esempio, le richieste indirizzate alla route che inizia con "/admin" saranno filtrate da isLoggedIn e isAdmin prima di poter essere gestite dal controller, in quanto accessibili unicamente da utenti loggati che siano anche amministratori del sistema.

Capitolo 5

Validazione

Terminato lo sviluppo secondo le specifiche descritte nei precedenti capitoli, l'applicazione è stata sottoposta a un esperimento di validazione preliminare. Scopo dell'esperimento è stato verificare il corretto comportamento dell'applicazione e raccogliere metriche [1] relative alle sue prestazioni.

5.1 Preparazione dei dati

La validazione preliminare è stata ideata e portata avanti senza il coinvolgimento di studenti, simulando soluzioni da loro prodotte che potessero risultare verosimili (Tabella 5.1). Si è scelta come punto di partenza la soluzione corretta di un laboratorio di PO risalente all'anno accademico 2021/2022. A partire da essa sono state generate più variazioni, iniettando al suo interno zero o più bug in ognuna di esse e aggiungendo test che potessero trovare zero o più bug nelle soluzioni altrui. Inoltre, sono state prodotte delle soluzioni che fallissero in fase di compilazione, validazione sulla soluzione ideale e sulla verifica del numero massimo di test consegnabili. Le soluzioni sono state caricate su repository GitLab appositamente creati.

La soluzione corretta presa come base ha assunto il ruolo di soluzione ideale. Anch'essa è stata caricata su un repository GitLab.

[1] Le metriche raccolte sono strettamente dipendenti dall'hardware su cui è stato effettuato l'esperimento.

Nickname	N° bug	N° test
validation_no_compile	1	5
validation_no_ideal	3	2
validation_no_max	0	8
validation_a	0	7
validation_b	4	7
validation_c	1	5
validation_d	3	3

Tabella 5.1: Soluzioni preparate per la validazione.

5.2 L'esperimento

Sulla base dei dati preparati sono state ideate e portate avanti due sessioni di laboratorio, con differente numero di utenti coinvolti e caratteristiche delle soluzioni. Entrambe le sessioni sono state simulate nel medesimo contesto hardware.

Come passaggio preliminare, tutti gli studenti associati alle soluzioni preparate sono stati registrati nel sistema (per mezzo della sezione "Registrazione" del frontend per giocatori) con i nickname di Tabella 5.1.

5.2.1 Prima sessione

La prima sessione inizia con l'apertura del laboratorio ad opera dell'amministratore. Quest'ultimo effettua l'accesso all'applicazione (tramite il frontend per amministratori) e crea un nuovo laboratorio dalla sezione "Labs". I campi scelti sono ininfluenti ai fini dell'esperimento, ad eccezione dell'URL del repository contenente la soluzione ideale e il numero massimo di test che ogni studente può consegnare (7, in questo caso).

Prendendo come riferimento la Tabella 5.1, solo tre studenti (validation_a, validation_b e validation_c) hanno preso parte al laboratorio, inserendo l'URL del repository contenente la propria soluzione nell'apposito form della sezione "Labs".

Forzando il termine del laboratorio è stato avviato il processo finale di verifica e assegnazione dei punteggi, che è stato analizzato a livello di correttezza dei risultati ottenuti (Tabella 5.2) e tempi di esecuzione delle varie fasi (Tabella 5.3).

Nickname	Passa la fase di verifica	Punti verifica	N° test propri falliti	N° test altrui passati	Punteggio finale
validation_a	Sì	1	5	12	18
validation_b	Sì	1	1	6	8
validation_c	Sì	1	2	12	15

Tabella 5.2: Punteggi relativi alla prima sessione.

	Filtraggio	**validation_a**	**validation_b**	**validation_c**
validation_a	115400	69883	62244	68880
validation_b	124846	64917	64253	72053
validation_c	114942	67300	69116	66574

Tabella 5.3: Tempi di esecuzione delle principali fasi della prima sessione (espressi in millisecondi).

La Tabella 5.3 riporta i tempi di esecuzione (espressi in millisecondi) delle fasi considerate più onerose per quanto riguarda il tempo di calcolo richiesto. In particolare:

- la colonna "Filtraggio" indica il tempo impiegato ad eseguire le tre verifiche iniziali (compilazione, soluzione ideale e numero di test);
- una casella di tipo [x][x] indica il tempo impiegato dall'esecuzione dei test dello studente sulla propria soluzione, in modo da ricavare i dati relativi alla copertura;
- una casella di tipo [x][y] indica il tempo impiegato per l'esecuzione dei test dell'utente x sulla soluzione dell'utente y.

Complessivamente è stato registrato un tempo di esecuzione della fase finale (dal forzamento del suo avvio alla sua conclusione) di 969570 millisecondi. È stata inoltre calcolata correttamente la copertura dei propri test sulle proprie classi e, al termine della simulazione di laboratorio, sono state correttamente aggiornate le classifiche (quella relativa al laboratorio appena concluso e quella globale) e le percentuali di completamento degli obiettivi di ogni utente.

5.2.2 Seconda sessione

La seconda sessione inizia con l'apertura del nuovo laboratorio (anch'esso con un numero massimo di 7 test consegnabili) da parte dell'amministratore, come avvenuto nella sessione precedente. Tutti gli utenti indicati in Tabella 5.1 hanno preso parte al laboratorio, caratterizzandolo con le seguenti tipologie di soluzione:

- 1 soluzione con un errore di sintassi;
- 1 soluzione con un errore nei test;
- 1 soluzione contenente 8 test (uno in più del numero massimo consentito per il laboratorio);

- 4 soluzioni corrette dal punto di vista formale.

Nickname	**Passa la fase di verifica**	**Punti verifica**	**N° test propri falliti**	**N° test altrui passati**	**Punteggio finale**
validation_no_compile	No	0	0	0	0
validation_no_ideal	No	0	0	0	0
validation_no_max	No	0	0	0	0
validation_a	Sì	1	8	15	24
validation_b	Sì	1	4	8	13
validation_c	Sì	1	4	14	19
validation_d	Sì	1	2	11	14

Tabella 5.4: Punteggi relativi alla seconda sessione

	Filtraggio	**validation_a**	**validation_b**	**validation_c**	**validation_d**
validation_no_compile	38669	-	-	-	-
validation_no_ideal	119836	-	-	-	-
validation_no_max	109014	-	-	-	-
validation_a	121834	64089	74654	58803	64917
validation_b	116814	61640	71501	69763	66045
validation_c	110149	81050	63824	58051	61580
validation_d	122967	66085	68652	65268	67338

Tabella 5.5: Tempi di esecuzione delle principali fasi della seconda sessione (espressi in millisecondi)

Come si può evincere da Tabella 5.4, le soluzioni degli utenti validation_no_compile, validation_no_ideal e validation_no_max non hanno preso parte al calcolo del punteggio finale, poiché correttamente filtrate: la prima non superava la verifica di compilazione, i test della seconda fallivano sulla soluzione ideale e la terza presentava un numero eccessivo di test rispetto al limite consentito. Tabella 5.5 può essere letta in maniera analoga a quella relativa al suo corrispettivo della prima sessione. In particolare si può notare come delle prima tre soluzioni sia stato riportato unicamente il tempo di filtraggio, in quanto nessuna delle tre supera i test di verifica per i motivi precedentemente illustrati.

È stato registrato un tempo totale di esecuzione del laboratorio di 1812957 millisecondi. Anche in questo caso, in seguito alla conclusione del laboratorio sono state aggiornate le classifiche e le percentuali di completamento degli obiettivi per ogni utente.

Al termine di entrambe le sessioni è stato possibile consultare – per ogni utente – l'esito dei laboratori e gli obiettivi raggiunti tramite il frontend per giocatori. L'amministratore ha potuto infine scaricare tutti i tipi di report illustrati nel precedente capitolo.

5.3 Analisi dei risultati

Lo scopo che ci si è posti con la validazione preliminare è quello di verificare che il sistema implementasse correttamente tutte le funzionalità descritte nei precedenti paragrafi e ottenere primi dati circa le sue performance.

Dal punto di vista dei risultati, il sistema si è comportato in maniera conforme alle aspettative, portando a termine l'intero processo senza errori in entrambe le simulazioni di laboratorio.

Per quanto riguarda le performance, le novità introdotte nel processo di svolgimento dei laboratori impediscono di ricavare dati probabilistici utili a effettuare stime sulla durata media di tutta la fase di verifica e calcolo dei punteggi (che, come spiegato precedentemente, sono i due punti del processo che segue il termine di un laboratorio che influiscono più pesantemente sulla sua durata). Per effettuare tali previsioni sarebbe stato necessario conoscere, ad esempio, la probabilità che uno studente consegni soluzioni che non superino i controlli che precedono il calcolo dei punteggi: soluzioni "errate" richiederebbero un tempo drasticamente inferiore ad altre, in quanto verrebbero subito filtrate.

È tuttavia possibile ricavare due dati utili a stimare l'intervallo temporale richiesto per l'esecuzione della fase di verifica e calcolo dei punteggi:

- il caso migliore, quello in cui tutte le soluzioni vengono filtrate;
- il caso peggiore, quello in cui tutte le soluzioni superano i controlli e partecipano al calcolo dei punteggi.

Per quanto riguarda il primo, sono stati presi in considerazione i risultati ottenuti nella seconda sessione da validation_no_compile (a cui è stato assegnato indice 1), validation_no_ideal (indice 2) e validation_no_max (indice 3), gli unici a non superare la fase di filtraggio. Si è ipotizzato inoltre che le tre cause di filtraggio (errore di sintassi, test non validi e superamento del numero massimo di test) si possano verificare in maniera equiprobabile. Si è quindi ricavato il tempo medio di filtraggio per ogni soluzione come:

$$\frac{\sum_{i=1}^{n} t_i}{n}$$

dove n è il numero di soluzioni considerate (3) e t(i) è il tempo impiegato per filtrare le tre soluzioni, rispettivamente. Si è così ottenuto un valore di 89173 millisecondi. Ipotizzando un totale di 100 partecipanti, il tempo impiegato per filtrare tutte le soluzioni sarebbe quindi di circa 149 minuti.

Per quanto riguarda il caso peggiore, si è proceduto a stimare separatamente il tempo totale dei due laboratori, considerando anche in questo caso un totale di 100 partecipanti e ignorando i casi di soluzioni che non superano la fase di filtraggio. Il tempo medio di esecuzione di un laboratorio è stato calcolato riprendendo la formula precedente (e assegnando indici alle soluzioni in maniera analoga), con – in questo caso – n pari a 3 nel caso del primo laboratorio e 4 nel secondo e con t(i) calcolato come:

$$t_i = t_{fi} + \sum_{j=1}^{n} \tau_j$$

dove t(fi) è il tempo di filtraggio della soluzione i-esima e tau(j) è il tempo impiegato a eseguire i test della soluzione i-esima su quella j-esima (quando i e j coincidono si tratta del caso di esecuzione dei propri test sulla propria soluzione per ricavare la copertura).

Per il primo laboratorio è stato stimato un tempo di esecuzione medio di 320136 millisecondi, mentre per il secondo di 383756 millisecondi: rispettivamente circa 9 e 10 ore considerando un totale di 100 partecipanti, per una media di 9.78 ore.

Si può dunque notare come anche nel caso peggiore il tempo richiesto per portare a termine tutto il processo considerato sia ragionevole, in quanto tutte le azioni considerate non richiederanno sincronia, ma potranno essere eseguite da un server nel periodo che intercorre tra il termine di un laboratorio e l'inizio del successivo.

Capitolo 6

Conclusioni

Questo lavoro di tesi ruota attorno alla progettazione e all'implementazione di Unit Brawl, un'applicazione con integrati elementi di gamification per favorire l'attività dello unit testing in ambito didattico.

L'applicazione consiste in più round durante i quali gli studenti sviluppano programmi Java con i rispettivi unit test, che verranno eseguiti sulle soluzioni degli avversari. Alla fine di ogni round vengono calcolati punteggi che contribuiscono all'aggiornamento di una classifica generale e che possono essere convertiti in monete virtuali per acquistare avatar.

Scopo di Unit Brawl è quello di avvicinare gli studenti all'ambito dello unit testing, sfruttando la gamification in modo da rendere questa attività più appagante e coinvolgente. L'obiettivo finale che ci si è posti di raggiungere è quello di aumentare l'interesse degli studenti verso l'argomento e – conseguentemente – la qualità del loro apprendimento di concetti ad esso correlati.

È stata scelta la competizione come motore principale attorno al quale ideare l'esperienza; ciò nonostante, sono stati integrati elementi quali avatar e obiettivi in modo da fornire scopi complementari alla competizione stessa. Così facendo, si è evitato di generalizzare il concetto di apprendimento, cercando di intercettare quante più sfumature possibili.

In seguito a una valutazione preliminare, il sistema si è comportato coerentemente rispetto ai risultati attesi, garantendo sicurezza circa la stabilità dell'attuale implementazione.

6.1 Limiti

Indipendentemente dall'esito della valutazione preliminare, Unit Brawl presenta alcuni limiti che verranno illustrati in seguito.

Il primo riguarda il file di Continuous Integration di GitLab, asset vulnerabile in quanto visibile e modificabile da ciascuno studente relativamente al proprio repository. Si individuano quindi minacce su due fronti: la possibilità di inviare false informazioni al server e quella di leggere informazioni nascoste. Per la precisione, la prima è già stata mitigata, in quanto ogni informazione proveniente da GitLab viene ritenuta inaffidabile ed è sfruttata unicamente per fornire prime e parziali indicazioni all'utente. Tutti i controlli definitivi vengono effettuati dal backend in fase di calcolo finale dei punteggi. Una possibile soluzione per la seconda sarebbe posporre i controlli sulla soluzione ideale a tempo di verifica finale, rimuovendo quindi qualsiasi riferimento ad essa da GitLab e racchiudendolo unicamente nel backend.

Un'altra limitazione che coinvolge GitLab riguarda l'associazione fra l'utente conosciuto dal server e quello legato alla pipeline di CI: per renderlo possibile è necessario che lo studente si registri alla piattaforma con lo stesso username usato per accedere a GitLab, con tutte le complicanze del caso derivanti da errori umani. Una possibile soluzione sarebbe implementare un meccanismo di login unificato (che sfrutti ad esempio le credenziali istituzionali in possesso di ogni studente o quelle dello stesso GitLab) in modo da delegare l'aspetto del login a servizi esterni all'applicazione stessa.

Volgendo lo sguardo alla gamification, è stato individuato un caso di possibile aggiramento delle regole ritenuto meritevole di approfondimento. Al centro della tematica vi sono i test cloni, ovvero test diversi ma il cui scopo è quello di rilevare lo stesso bug su una soluzione nemica. Questo caso bilancia intrinsecamente vantaggi e svantaggi: da un lato, se i test cloni andassero a buon fine, allora l'utente riuscirebbe a ottenere molteplici punti grazie all'individuazione di un singolo bug; dall'altro, se i test cloni non riuscissero a individuare il bug, risulterebbero essere uno spreco di risorse che l'utente avrebbe potuto sfruttare per coprire più casistiche.

Per quanto riguarda la generalizzabilità dell'applicazione presentata, questa dipende imprescindibilmente dallo specifico linguaggio e fase di test. Si ritiene l'approccio battle-royale (e le meccaniche di Unit Brawl ad esso legate) estendibile a più tipologie di attività di verifica e non solo allo unit testing black-box e white-box.

Infine, nonostante i risultati positivi derivanti dalla validazione preliminare descritta in precedenza, Unit Brawl deve ancora essere sottoposto a una fase di verifica sul campo. Per la scelta di quali meccaniche di gamification adottare si è attinto da quelle più conosciute e apprezzate in letteratura, ma – come varie fonti fanno notare – l'esito della loro applicazione varia di ambito in ambito e dipende strettamente dall'effettiva messa in opera. Si è quindi messo in conto che alcune meccaniche possano portare a risultati opposti a quelli ricercati, motivo per il quale una validazione che coinvolga gli utenti finali è stata identificata come prossima attività da portare avanti. Scopo di quest'ultima sarà quello di ottenere dati che possano dare una misura dell'efficacia delle meccaniche di gamification adottate.

6.2 Sviluppi futuri

Come anticipato nel precedente paragrafo, la prossima attività in programma sarà quella di validare l'applicazione per mezzo di un esperimento che coinvolga gli studenti. Unit Brawl verrà introdotta nel corso di PO e ne supporterà alcuni laboratori svolti durante il semestre. Così facendo sarà possibile ricavare dati chiari circa l'efficacia dell'implementazione attuale.

Inoltre, si prevede di implementare ulteriori meccaniche di gamification in modo da arricchire e approfondire l'esperienza. Anche in questo caso si è deciso di attingere dalla letteratura, individuando vari casi positivi di integrazione di uno strato narrativo all'interno della piattaforma tramite termini o metafore grafiche che possano astrarre le attività portate avanti dagli studenti. Benefici derivanti dall'utilizzo di metafore per visualizzare metriche o problemi da risolvere vengono portati alla luce da Balogh et al. [4], in cui gli autori sfruttano parti di una città virtuale (giardini, costruzioni, ecc.) per rappresentare tali elementi.

Appendice A

Laboratorio di validazione

La presente appendice riporta i requisiti della traccia di laboratorio scelta come punto di partenza per l'esperimento di validazione preliminare.

A.1 Requisiti

Progettare ed implementare un programma che possa gestire corsi, docenti e studenti di un ateneo. Tutte le classi devono appartenere al package *university*. Il programma interagisce con i propri clienti attraverso la classe di facciata *University*.

A.1.1 R1. Ateneo

La classe principale è *University* che riceve, come parametro del costruttore, il nome dell'ateneo.

Il nome dell'ateneo è leggibile tramite il metodo getter *getName()*.

È possibile definire il nome del rettore di un ateneo tramite il metodo *setRector()* che riceve come parametri nome e cognome del rettore.

Il metodo getter *getRector()* restituisce nome e cognome del rettore concatenati e separati da uno spazio (" ").

A.1.2 R2. Studenti

È possibile inserire le informazioni relative ad un nuovo studente tramite il metodo *enroll()* della classe *University*, che riceve come parametri il nome ed il cognome dello studente; il metodo restituisce il numero di matricola che è stato assegnato allo studente. I numeri di matricola vengono assegnati, in maniera progressiva per ciascun ateneo a partire dal numero 10000. Quindi il primi studente iscritto ad ogni ateneo avrà matricola 10000

Per ottenere le informazioni relative ad uno studente si utilizza il metodo *student()* che riceve come parametro la matricola e restituisce una stringa composta da numero di matricola, nome e cognome separati da spazi, es. "10000 Giuseppe Verdi".

Si assuma che ciascun ateneo non possa contenere più di 1000 studenti.

A.1.3 R3. Insegnamenti

Per definire un nuovo insegnamento si utilizza il metodo *activate()* che riceve come parametri il titolo del corso e il nome del docente titolare. Il metodo restituisce un intero che corrisponde al codice del corso. I codici vengono assegnati progressivamente a partire da 10.

Per conoscere le informazioni relative ad un corso si usa il metodo *course()* che riceve come parametro il codice del corso e restituisce una stringa contenente codice, titolo e titolare del corso, separati da virgole, es. "10,Programmazione a Oggetti,James Gosling".

Si assuma che ciascun ateneo non possa attivare più di 50 insegnamenti.

A.1.4 R4. Iscritti agli insegnamenti

Gli studenti possono essere iscritti agli insegnamenti tramite il metodo *register()* che riceve come parametro la matricola dello studente ed il codice dell'insegnamento.

Per ottenere l'elenco degli iscritti ad un insegnamento è disponibile il metodo *listAttendees()* che riceve come parametro il codice dell'insegnamento e restituisce una stringa contenente l'elenco degli studenti iscritti.

Gli studenti compaiono uno per riga (le righe sono terminate da un a-capo) secondo il formato descritto al punto R2.

Data la matricola di uno studente, tramite il metodo *studyPlan()*, è possibile conoscere l'elenco degli insegnamenti a cui è iscritto, gli insegnamenti sono descritti come al punto precedente.

Si assuma che ciascuno studente non possa essere iscritto a più di 25 insegnamenti e che un insegnamento non possa avere più di 100 iscritti.

Appendice B

API

Di seguito verranno elencate le API esposte dal backend, suddivise per route. Per ognuna di esse è riportato l'URL di riferimento, il tipo di metodo HTTP, una descrizione, un esempio di body della richiesta (facoltativo), i possibili status della risposta e un esempio di body della risposta (facoltativo).

B.1 API admin

Elenco di API accessibili solo da utenti di tipo amministratore.

URL: /admin/labs
Metodo HTTP: GET
Descrizione: restituisce una lista di tutti i laboratori (ognuno con l'url della soluzione ideale)
Body della richiesta: -
Status della risposta: 200 (ok), 500 (errore nel database)
Body della risposta:

```
[
    {
        "id": 1,
        "name": "Lab 1",
        "deadline": "31-11-2022",
        "trace": "trace 1",
        "expired": false,
        "leaderboard": [
            {
                "labName": "Lab 1",
                "completed": true,
```

```
                "points": 5,
                "position": 1,
                "username": "user_1",
                "userAvatarLink": "link"
            },
            {
                "labName": "Lab 1",
                "completed": true,
                "points": 3,
                "position": 2,
                "username": "user_2",
                "userAvatarLink": "link"
            }
        ],
        "testMaxNumber": 12,
        "linkToIdealSolution": "repo 1"
    },
    {
        "id": 2,
        "name": "Lab 2",
        "deadline": "26-11-2022",
        "trace": "Trace 2",
        "expired": true,
        "leaderboard":
        [
            {
                "labName": "Lab 2",
                "completed": true,
                "points": 6,
                "position": 1,
                "username": "user",
                "userAvatarLink": "link"
            }
        ],
        "testMaxNumber": 1,
        "linkToIdealSolution": "link"
    }
]
```


URL: /admin/labs
Metodo HTTP: POST
Descrizione: crea un nuovo laboratorio attivo

Body della richiesta:

```
{
    "id": 2, //ignorato dal server
    "name": "Lab 2",
    "deadline": "26-11-2022",
    "trace": "Trace 2",
    "expired": true,
    "leaderboard": null,
    "testMaxNumber": 6,
    "linkToIdealSolution": "link"
}
```

Status della risposta: 200 (ok), 500 (errore nel database)
Body della risposta: -

URL: /admin/labs/active
Metodo HTTP: GET
Descrizione: restituisce il laboratorio attivo (se c'è)
Body della richiesta: -
Status della risposta: 200 (ok), 500 (errore nel database), 404 (se non ci sono laboratori attivi)
Body della risposta:

```
{
    "id": 1,
    "name": "Lab 1",
    "deadline": "31-11-2022",
    "trace": "trace 1",
    "expired": false,
    "leaderboard": []
    "testMaxNumber": 12,
    "linkToIdealSolution": "repo 1"
}
```

URL: /admin/labs/:id/leaderboard
Metodo HTTP: GET
Descrizione: restituisce la leaderboard del laboratorio con id <id>
Body della richiesta: -
Status della risposta: 200 (ok), 500 (errore nel database)
Body della risposta:

```
[
```

```
    {
        "labName": "Lab 1",
        "completed": true,
        "points": 5,
        "position": 1,
        "username": "user_1",
        "userAvatarLink": "link1"
    },
    {
        "labName": "Lab 2",
        "completed": true,
        "points": 2,
        "position": 2,
        "username": "user_2",
        "userAvatarLink": "link2"
    }
]
```


URL: /admin/labs
Metodo HTTP: PUT
Descrizione: aggiorna un laboratorio
Body della richiesta: -
Status della risposta: 200 (ok), 500 (errore nel database)
Body della risposta:


```
{
    "id": 1,
    "name": "Lab 1",
    "deadline": "31-11-2022",
    "trace": "trace 1",
    "expired": false,
    "leaderboard": [
        {
        "labName": "Lab 1",
        "completed": true,
        "points": 5,
        "position": 1,
        "username": "user_1",
        "userAvatarLink": "link"
        },
     {
            "labName": "Lab 1",
```

```
                "completed": true,
                    "points": 3,
                "position": 2,
                "username": "user_2",
                "userAvatarLink": "link"
            }
        ],
        "testMaxNumber": 12,
        "linkToIdealSolution": "repo 1"
    }
```

URL: /admin/labs/:id
Metodo HTTP: DELETE
Descrizione: elimina il laboratorio con id <id>
Body della richiesta: -
Status della risposta: 200 (ok), 500 (errore nel database)
Body della risposta: -

URL: /admin/labs/stop/:id
Metodo HTTP: POST
Descrizione: interrompe il laboratorio con id <id>
Body della richiesta: -
Status della risposta: 200 (ok), 500 (errore nel database), 404 (laboratorio non esistente)
Body della risposta: -

URL: /admin/labs/:id
Metodo HTTP: GET
Descrizione: restituisce il laboratorio con id <id>
Body della richiesta: -
Status della risposta: 200 (ok), 500 (errore nel database), 404 (laboratorio non esistente)
Body della risposta:

```
    {
        "id": 1,
        "name": "Lab 1",
        "deadline": "31-11-2022",
        "trace": "trace 1",
        "expired": false,
        "leaderboard": [
            {
```

```
            "labName": "Lab 1",
            "completed": true,
            "points": 5,
            "position": 1,
            "username": "user_1",
            "userAvatarLink": "link"
            },
         {
                "labName": "Lab 1",
                "completed": true,
                    "points": 3,
                "position": 2,
                "username": "user_2",
                "userAvatarLink": "link"
            }
        ],
        "testMaxNumber": 12,
        "linkToIdealSolution": "repo 1"
    }
```


URL: /admin/labs/:id/players/count
Metodo HTTP: GET
Descrizione: restituisce il numero di giocatori iscritti al laboratorio con id <id>
Body della richiesta: -
Status della risposta: 200 (ok), 500 (errore nel database), 404 (laboratorio non esistente)
Body della risposta: 5

URL: /admin/avatars
Metodo HTTP: GET
Descrizione: restituisce la lista di avatar
Body della richiesta: -
Status della risposta: 200 (ok), 500 (errore nel database)
Body della risposta:


```
    [
    {
        "id": 1,
        "name": "a1",
        "description": "a1",
        "price": 1,
        "image_path": "img path avatar 1"
```

```
},
{
    "id": 2,
    "name": "a2",
    "description": "a2",
    "price": 3.6,
    "image_path": "img path avatar 2"
}
]
```

URL: /admin/avatars/:id
Metodo HTTP: DELETE
Descrizione: elimina l'avatar con id <id>
Body della richiesta: -
Status della risposta: 200 (ok), 500 (errore nel database)
Body della risposta: -

URL: /admin/avatars Metodo HTTP: PUT Descrizione: aggiorna l'avatar Body della richiesta:

```
{
    "id": 1,
    "name": "a1",
    "description": "d1",
    "price": 1,
    "imagePath": "img path avatar 1"
}
```

Status della risposta: 200 (ok), 500 (errore nel database) Body della risposta: -

URL: /admin/avatars
Metodo HTTP: POST
Descrizione: crea un nuovo avatar
Body della richiesta:

```
{
"id": 1, //ignorato dal server
    "name": "a1",
    "description": "d1",
    "price": 1,
    "imagePath": "img path avatar 1"
}
```

Status della risposta: 200 (ok), 500 (errore nel database)
Body della risposta: -
URL: /admin/reports/general
Metodo HTTP: GET
Descrizione: ottiene un report generale sul sistema
Body della richiesta: -
Status della risposta: 200 (ok), 500 (errore nel database)
Body della risposta:

```
{
    "labTotalNumber": 4,
    "userTotalNumber": 3,
    "avgParticipantsPerLab": 0.75,
    "labsWithoutParticipantsCount": 2
}
```

URL: /admin/reports/labs
Metodo HTTP: GET
Descrizione: ottiene un report sui laboratori
Body della richiesta: -
Status della risposta: 200 (ok), 500 (errore nel database)
Body della risposta:

```
[
{
    "id": 0,
    "participants": 0,
    "participantsPercentage": 0,
    "avgPoints": 0
},
{
    "id": 1,
    "participants": 2,
    "participantsPercentage": 66.66666666666667,
    "avgPoints": 3.5
}
]
```

URL: /admin/reports/users
Metodo HTTP: GET
Descrizione: ottiene un report sugli utenti
Body della richiesta: -
Status della risposta: 200 (ok), 500 (errore nel database)

Body della risposta:


```
  [
  {
      "id": 1,
      "nickname": "user 1",
      "fullName": "Mario Rossi",
      "labsAttendedCount": 1,
      "labsAttendedPercentage": 25,
      "avgPoints": 5,
      "bestPosition": 2
  },
  {
      "id": 2,
      "nickname": "antmat99",
      "fullName": "Luigi Verdi",
      "labsAttendedCount": 2,
      "labsAttendedPercentage": 50,
      "avgPoints": 4,
      "bestPosition": 1
  }
]
```


URL: /admin/leaderboard
Metodo HTTP: GET
Descrizione: restituisce la leaderboard globale
Body della richiesta: -
Status della risposta: 200 (ok), 500 (errore nel database)
Body della risposta:


```
  [
      {
          "labName": "",
          "completed": true,
          "points": 100,
          "position": 1,
          "username": "user_1",
          "userAvatarLink": "link_1"
      },
      {
          "labName": "",
          "completed": true,
```

```
            "points": 50,
            "position": 2,
            "username": "user_2",
            "userAvatarLink": "link_2"
        },
        {
            "labName": "",
            "completed": true,
            "points": 10,
            "position": 3,
            "username": "user_3",
            "userAvatarLink": "link_3"
        }
    ]
```

URL: /admin/reports/userlabs
Metodo HTTP: GET
Descrizione: ottiene un report su laboratori e utenti
Body della richiesta: -
Status della risposta: 200 (ok), 500 (errore nel database)
Body della risposta:

```
[
    {
        "user_id": 1,
        "lab_id": 4,
        "points": 133,
        "position": 1,
        "repository": "https://gitlab.com/user/project.git",
        "coverage_percentage": 65,
        "test_failed_on_enemies": 32
        "test_passed_on_own": 54
    },
    {
        "user_id": 3,
        "lab_id": 2,
        "points": 13,
        "position": 56,
        "repository": "https://gitlab.com/user2/project2.git",
        "coverage_percentage": 14,
        "test_failed_on_enemies": 3
        "test_passed_on_own": 69
```

```
    }
]
```

B.2 API login

Elenco di API coinvolte nelle fasi di registrazione e autenticazione.

URL: /sessions
Metodo HTTP: POST
Descrizione: login
Body della richiesta:

```
{
    "username":"user",
    "password":"Password1."
}
```

Status della risposta: 200 (ok), 500 (errore nel database), 401 (unauthorized)
Body della risposta: -

URL: /sessions/register
Metodo HTTP: POST
Descrizione: registrazione
Body della richiesta:

```
{
    "user":{
        "nickname":"as",
        "password":"password",
        "name":"new name",
        "surname":"new surname",
        "email":"a@a.it"
    }
}
```

Status della risposta: 200 (ok), 500 (errore nel database), 400 (in caso di email, nickname già esistente o nickname uguale a "admin")
Body della risposta: -

URL: /sessions/current
Metodo HTTP: GET
Descrizione: restituisce informazioni sull'utente correntemente loggato

Body della richiesta: -
Status della risposta: 200 (ok), 500 (errore nel database), 401 (unauthorized)
Body della risposta:

```
{

    "id": 1,
    "nickname": "nick",
    "password": "123456",
    "name": "Antonio",
    "surname": "Mat",
    "total_points": 23,
    "money": 21.6,
    "avatar_selected_id": 1,
    "email": "a@a.it"
}
```

URL: /sessions/current
Metodo HTTP: DELETE
Descrizione: logout
Body della richiesta: -
Status della risposta: 200 (ok), 500 (errore nel database), 401 (unauthorized)
Body della risposta: -

B.3 API generali

Elenco di API accessibili da utenti generici.

URL: /avatars/shop
Metodo HTTP: POST
Descrizione: processo di acquisto di uno o più avatar
Body della richiesta:

```
{
    "avatarIdList": [1,2,7,3]
}
```

Status della risposta: 200 (ok), 500 (errore nel database), 400 (se l'utente non ha abbastanza soldi o ha già l'avatar)
Body della risposta: -

URL: /labs

Metodo HTTP: GET

Descrizione: restituisce la lista di tutti i laboratori (senza il link alla repository della soluzione). Se l'utente non ha partecipato al laboratorio, userResult e username saranno undefined.

Body della richiesta: [1,2,7,3]

Status della risposta: 200 (ok), 500 (errore nel database), 404 (se non ci sono laboratori attivi), 400 (se l'utente non ha abbastanza soldi o ha già l'avatar)

Body della risposta:

```
[
Lab {
id: 27,
name: 'Lab 1',
deadline: '30-11-2022',
trace: 'trace 1',
expired: true,
leaderboard: [
    {
    "labName": "Lab 1",
    "completed": true,
    "points": 5,
    "position": 1,
    "username": "user_1",
    "userAvatarLink": "link"
    },
    {
        "labName": "Lab 1",
        "completed": true,
        "points": 3,
        "position": 2,
        "username": "user_2",
        "userAvatarLink": "link"
    }
],
userResult: Result {
  labName: 'Lab 1',
  completed: true,
  points: 130,
  position: 1,
  username: undefined,
  userAvatarLink: undefined
},
```

```
    username: 'username'
  },
  Lab {
    id: 28,
    name: 'Lab 2 new',
    deadline: '30-11-2022',
    trace: 'trace 2',
    expired: true,
    leaderboard: [
        {
        "labName": "Lab 1",
        "completed": true,
        "points": 5,
        "position": 1,
        "username": "user_1",
        "userAvatarLink": "link"
        },
        {
            "labName": "Lab 1",
            "completed": true,
            "points": 3,
            "position": 2,
            "username": "user_2",
            "userAvatarLink": "link"
        }
     ],
    userResult: {
      labName: 'Lab 2 new',
      completed: true,
      points: 24,
      position: 2,
      username: undefined,
      userAvatarLink: undefined
    },
    username: 'username'
  }
]
```

URL: /leaderboard/:quantity
Metodo HTTP: GET
Descrizione: restituisce le prime <quantity> righe della leaderboard globale
Body della richiesta: -

Status della risposta: 200 (ok), 500 (errore nel database)
Body della risposta:


```
[
    {
        "labName": "",
        "completed": true,
        "points": 100,
        "position": 1,
        "username": "user_1",
        "userAvatarLink": "link_1"
    },
    {
        "labName": "",
        "completed": true,
        "points": 50,
        "position": 2,
        "username": "user_2",
        "userAvatarLink": "link_2"
    },
    {
        "labName": "",
        "completed": true,
        "points": 10,
        "position": 3,
        "username": "user_3",
        "userAvatarLink": "link_3"
    }
]
```


URL: /leaderboard/region
Metodo HTTP: GET
Descrizione: restituisce una leaderboard parziale, contenente informazioni sull'utente loggato e su quelli in posizione precedente e successiva a lui.
Body della richiesta: -
Status della risposta: 200 (ok), 500 (errore nel database)
Body della risposta:


```
[
  {
    id: 24,
    points: 4000,
```

```
    avatar_selected_id: 0,
    userAvatarLink: 'link_1',
    position: 1,
    username: 'before'
  },
  {
    id: 26,
    points: 3990,
    avatar_selected_id: 0,
    userAvatarLink: 'link_2',
    position: 2,
    username: 'user'
  },
  {
    id: 25,
    points: 3975,
    avatar_selected_id: 0,
    userAvatarLink: 'link_3',
    position: 3,
    username: 'after'
  }
]
```

URL: /users/results
Metodo HTTP: GET
Descrizione: restituisce la lista di risultati dell'utente loggato
Body della richiesta: -
Status della risposta: 200 (ok), 500 (errore nel database)
Body della risposta:

```
[
{
    "nickname": "nick",
    "labName": "Lab 1",
    "points": 39,
    "position": 43
},
{
    "nickname": "nick",
    "labName": "Lab 2",
    "points": 21,
    "position": 3
```

```
},
{
    "nickname": "nick",
    "labName": "Lab 3",
    "points": 311,
    "position": 4
}
]
```


URL: /users/avatars
Metodo HTTP: GET
Descrizione: restituisce la lista di avatar dell'utente loggato
Body della richiesta: -
Status della risposta: 200 (ok), 500 (errore nel database)
Body della risposta:


```
[
{
    "id": 1,
    "name": "Avatar 1",
    "description":"Avatar 1 description",
    "imagePath":"Link_avatar_1_image",
    "price": 34
},
{
    "id": 2,
    "name": "Avatar 2",
    "description":"Avatar 2 description",
    "imagePath":"Link_avatar_2_image",
    "price": 3
}
]
```


URL: /users/avatars/not
Metodo HTTP: GET
Descrizione: restituisce la lista di avatar che non sono in possesso dell'utente loggato
Body della richiesta: -
Status della risposta: 200 (ok), 500 (errore nel database)
Body della risposta:


```
[
{
```

```
    "id": 1,
    "name": "Avatar 1",
    "description":"Avatar 1 description",
    "imagePath":"Link_avatar_1_image",
    "price": 34
},
{
    "id": 2,
    "name": "Avatar 2",
    "description":"Avatar 2 description",
    "imagePath":"Link_avatar_2_image",
    "price": 3
}
]
```

URL: /users/avatars/selected
Metodo HTTP: GET
Descrizione: restituisce l'avatar scelto dall'utente come immagine del profilo
Body della richiesta: -
Status della risposta: 200 (ok), 500 (errore nel database)
Body della risposta:

```
  {
id: 0,
name: 'Defaulty',
description: 'Just your default look!',
price: 0,
image_path: 'https://avataaars.io/?avatarStyle=Circle'
  }
```

URL: /users/avatars/selected
Metodo HTTP: PUT
Descrizione: aggiorna l'avatar scelto dall'utente come immagine del profilo
Body della richiesta:

```
{
    "avatarId": 1
}
```

Status della risposta: 200 (ok), 500 (errore nel database)
Body della risposta: -

URL: /users/name

Metodo HTTP: GET
Descrizione: restituisce il nome dell'utente loggato
Body della richiesta: -
Status della risposta: 200 (ok), 500 (errore nel database)
Body della risposta: "Name"

URL: /users/surname
Metodo HTTP: GET
Descrizione: restituisce il cognome dell'utente loggato
Body della richiesta: -
Status della risposta: 200 (ok), 500 (errore nel database)
Body della risposta: "Surname"

URL: /users/nickname
Metodo HTTP: GET
Descrizione: restituisce il nickname dell'utente loggato
Body della richiesta: -
Status della risposta: 200 (ok), 500 (errore nel database)
Body della risposta: "Nickname"

URL: /users/achievements
Metodo HTTP: GET
Descrizione: restituisce la lista degli achievement dell'utente
Body della richiesta: -
Status della risposta: 200 (ok), 500 (errore nel database)
Body della risposta:


```
[
{
    "name": "Achievement 1 name",
    "description": "Achievement 1 description",
    "badge_path": "achievement_1_image_link",
    "completition_percentage": "30%",
    "code":"AVATAR_4"
},
{
    "name": "Achievement 2 name",
    "description": "Achievement 2 description",
    "badge_path": "achievement_2_image_link",
    "completition_percentage": "0%",
    "code":"JOIN_1"
```

```
},
]
```

URL: /users/achievements/fake
Metodo HTTP: GET
Descrizione: restituisce la lista degli achievement provvisori dell'utente riguardanti la coverage
Body della richiesta: -
Status della risposta: 200 (ok), 500 (errore nel database)
Body della risposta:

```
[
{
    "name": "Achievement 1 name",
    "description": "Achievement 1 description",
    "badge_path": "achievement_1_image_link",
    "completition_percentage": "30%",
    "code": "COVERAGE_50_1"
},
{
    "name": "Achievement 2 name",
    "description": "Achievement 2 description",
    "badge_path": "achievement_2_image_link",
    "completition_percentage": "0%",
    "code": "COVERAGE_70_1"
},
]
```

URL: /users/money
Metodo HTTP: GET
Descrizione: restituisce la quantità di monete dell'utente loggato
Body della richiesta: -
Status della risposta: 200 (ok), 500 (errore nel database)
Body della risposta: 43

URL: /users/achievements/quantity
Metodo HTTP: GET
Descrizione: restituisce la quantità di achievement dell'utente loggato completi al 100%
Body della richiesta: -
Status della risposta: 200 (ok), 500 (errore nel database)
Body della risposta: 4

URL: /users/labs
Metodo HTTP: GET
Descrizione: restituisce la lista di id dei laboratori a cui l'utente ha partecipato
Body della richiesta: -
Status della risposta: 200 (ok), 500 (errore nel database), 404 (utente non trovato)
Body della risposta:

```
[
    {
        "lab_id":27
    },
    {
        "lab_id":29
    }
]
```

URL: /labs/join
Metodo HTTP: POST
Descrizione: processo di iscrizione al laboratorio in corso. L'id dell'utente viene ricavato dall'utente loggato.
Body della richiesta:

```
{
    "repositoryLink": "link"
}
```

Status della risposta: 200 (ok), 500 (errore nel database), 400 (se non c'è nessun laboratorio in corso)
Body della risposta: -

URL: /users/labs/regionLeaderboard?labId=<labId>
Metodo HTTP: GET
Descrizione: restituisce la leaderboard contenente la posizione dell'utente nel lab <labId> e quelle immediatamente prima e dopo. L'id dell'utente viene ricavato dall'utente loggato.
Body della richiesta: -
Status della risposta: 200 (ok), 500 (errore nel database), 400 (labId è di un lab in corso)
Body della risposta:

```
[
    {
```

```
        "labName": "Lab 1",
        "completed": true,
        "points": 564,
        "position": 19,
        "username": "user_1",
        "userAvatarLink": "link"
        },
        {
            "labName": "Lab 1",
            "completed": true,
            "points": 20,
            "position": 456,
            "username": "user",
            "userAvatarLink": "link"
        },
        {
            "labName": "Lab 1",
            "completed": true,
            "points": 238,
            "position": 21,
            "username": "user_2",
            "userAvatarLink": "link"
        }
    ]
```


URL: /users/labs/active/joined
Metodo HTTP: GET
Descrizione: restituisce true se e solo se esiste un lab attivo e l'utente vi si è iscritto
Body della richiesta: -
Status della risposta: 200 (ok), 500 (errore nel database)
Body della risposta: true

URL: /users/labs/repositoryLink
Metodo HTTP: GET
Descrizione: aggiorna l'URL del repository dell'utente relativo al laboratorio in corso.
Body della richiesta: -
Status della risposta: 200 (ok), 500 (errore nel database)
Body della risposta:


```
  "link": "https://gitlab.com/user/project.git"
```

URL: /users/labs/repositoryLink
Metodo HTTP: GET
Descrizione: aggiorna l'URL del repository dell'utente relativo al laboratorio in corso.
Body della richiesta:

```
{
    "link": "https://gitlab.com/user/project.git"
}
```

Status della risposta: 200 (ok), 500 (errore nel database)
Body della risposta: -

URL: /gitlab/coverage/<nickname>
Metodo HTTP: POST
Descrizione: salva i dati della percentuale di copertura ottenuta a laboratorio in corso.
Body della richiesta:

```
{
    "link": "https://gitlab.com/user/project.git"
}
```

Status della risposta: 200 (ok), 500 (errore nel database)
Body della risposta: -

Bibliografia

[1] URL: https://getavataaars.com/.

[2] *$1.1 Trillion Impacted by Software Defects: A Testing Fail?* 2017. URL: https://web.archive.org/web/20200501113035/https://www.tricentis.com/blog/1-1-trillion-in-assets-impacted-by-software-defects-a-software-testing-fail.

[3] Cláuvin Almeida et al. «Negative effects of gamification in education software: Systematic mapping and practitioner perceptions». In: *Information and Software Technology* 156 (2023), p. 107142. ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2022.107142. URL: https://www.sciencedirect.com/science/article/pii/S0950584922002518.

[4] Gergo Balogh et al. «Using the City Metaphor for Visualizing Test-Related Metrics». In: *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. Vol. 2. 2016, pp. 17–20. DOI: 10.1109/SANER.2016.48.

[5] Kent Beck. *Test-Driven Development: By Example*.

[6] Moritz Beller et al. «Developer Testing in the IDE: Patterns, Beliefs, and Behavior». In: *IEEE Transactions on Software Engineering* 45.3 (2019), pp. 261–284. DOI: 10.1109/TSE.2017.2776152.

[7] Mike Cohn. *Succeeding with Agile: Software Development Using Scrum*. 1st. Addison-Wesley Professional, 2009.

[8] Sebastian Deterding et al. «c». In: *Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments*. MindTrek '11. Tampere, Finland: Association for Computing Machinery, 2011, 9–15. ISBN: 9781450308168. DOI: 10.1145/2181037.2181040. URL: https://doi.org/10.1145/2181037.2181040.

[9] Sebastian Elbaum et al. «Bug Hunt: Making Early Software Testing Lessons Engaging and Affordable». In: *29th International Conference on Software Engineering (ICSE'07)*. 2007, pp. 688–697. DOI: 10.1109/ICSE.2007.23.

[10] Igor Ernesto Ferreira Costa e Sandro Ronaldo Bezerra Oliveira. «The use of gamification to support the teaching-learning of software exploratory testing: an experience report based on the application of a framework». In: *2020 IEEE Frontiers in Education Conference (FIE)*. 2020, pp. 1–9. DOI: 10.1109/FIE44824.2020.9273943.

[11] N. Figol et al. «Application fields of gamification». In: *Amazonia Investiga* 10.37 (2021), pp. 93–100. DOI: 10.34069/AI/2021.37.01.9. URL: https://amazoniainvestiga.info/index.php/amazonia/article/view/1526.

[12] Gordon Fraser, Alessio Gambi e José Miguel Rojas. «Teaching Software Testing with the Code Defenders Testing Game: Experiences and Improvements». In: *2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. 2020, pp. 461–464. DOI: 10.1109/ICSTW50294.2020.00082.

[13] Juho Hamari, Jonna Koivisto e Harri Sarsa. «Does Gamification Work? – A Literature Review of Empirical Studies on Gamification». In: *2014 47th Hawaii International Conference on System Sciences*. 2014, pp. 3025–3034. DOI: 10.1109/HICSS.2014.377.

[14] «IEEE Standard Glossary of Software Engineering Terminology». In: *IEEE Std 610.12-1990* (1990), pp. 1–84. DOI: 10.1109/IEEESTD.1990.101064.

[15] Yu kai Chou. *Actionable Gamification: Beyond Points, Badges and Leaderboards*.

[16] Firas Khatib et al. «Crystal structure of a monomeric retroviral protease solved by protein folding game players». In: (2011). DOI: https://doi.org/10.1038/nsmb.2119.

[17] Matheus Marabesi e Ismar Frango Silveira. «Towards a Gamified Tool to Improve Unit Test Teaching». In: *2019 XIV Latin American Conference on Learning Technologies (LACLO)*. 2019, pp. 12–19. DOI: 10.1109/LACLO49268.2019.00013.

[18] MIT Massachusetts Institute of Technology. *Prototyping*. Accessed: 2022-11-15. 2018. URL: http://web.mit.edu/6.813/www/sp18/classes/11-prototyping/#reading_11_prototyping.

[19] Elvira G. Rincon-Flores et al. «Gamit! Interactive platform for gamification». In: *2022 IEEE Global Engineering Education Conference (EDUCON)*. 2022, pp. 10–13. DOI: 10.1109/EDUCON52537.2022.9766380.

[20] Karen Robson et al. «Is it all a game? Understanding the principles of gamification». In: *Business Horizons* 58.4 (2015), pp. 411–420. ISSN: 0007-6813. DOI: https://doi.org/10.1016/j.bushor.2015.03.006. URL: https://www.sciencedirect.com/science/article/pii/S000768131500035X.

[21] José Miguel Rojas e Gordon Fraser. «Code Defenders: A Mutation Testing Game». In: *2016 IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. 2016, pp. 162–167. DOI: 10.1109/ICSTW.2016.43.

[22] Philipp Straubinger e Gordon Fraser. «Gamekins: Gamifying Software Testing in Jenkins». In: *2022 IEEE/ACM 44th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. 2022, pp. 85–89. DOI: 10.1145/3510454.3516862.

[23] Dave Towey e Tsong Yueh Chen. «Teaching software testing skills: Metamorphic testing as vehicle for creativity and effectiveness in software testing». In: *2015 IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE)*. 2015, pp. 161–162. DOI: 10.1109/TALE.2015.7386036.