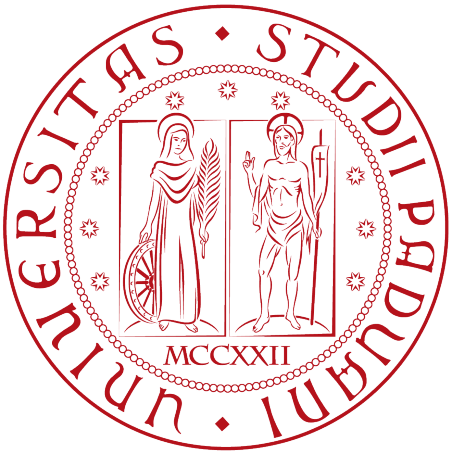
Università degli Studi di Padova

Dipartimento di Matematica

Corso di Laurea in Informatica



Refactoring di una piattaforma CMS: database MySQL, API REST ed interfaccia grafica

Tesi di laurea triennale

Relatore
Prof. Ombretta Gaggi

Laureando
Andrea Tombolato

Anno Accademico 2015-2016

Per aspera ad astra.

— Seneca

Sommario

Il presente documento descrive il lavoro svolto durante il periodo di stage, della durata di circa trecento ore, dal laureando Andrea Tombolato presso l'azienda Digital Accademia s.r.l.
L'obiettivo di tale attività era quello effettuare il refactoring del CMS utilizzato dall'azienda nell'ambito del progetto Plot. Il CMS si basa su un database MySQL ed è accessibile attraverso un'interfaccia web. Il refactoring è stato esteso anche al database e all'interfaccia web, è stata inoltre aggiunta la possibilità di accedere alle funzionalità del CMS attraverso delle API REST.
La struttura del CMS poggia sul framework Laravel mentre l'interfaccia web è stata ottenuta utilizzando un template Bootstrap.

Ringraziamenti

Innanzitutto vorrei esprimere la mia gratitudine alla Prof. Ombretta Gaggi, relatore della mia tesi, per l'aiuto ed il sostegno fornitomi durante la stesura di questo lavoro.

Desidero ringraziare con affetto mia sorella Beatrice ed i miei genitori, Germano e Paola, per tutto ciò che hanno fatto e continuano a fare per me. Senza di loro non avrei potuto intraprendere e portare a termine questo percorso. Più il tempo passa e più mi accorgo di quanto io sia fortunato ad avervi.

Un ringraziamento speciale va ai miei amici Tommaso, Alberto, Riccardo, Leonardo e Nicola per la loro pazienza. In molte occasioni ho dovuto abbandonarli a causa degli impegni relativi allo studio o al lavoro, ma loro non mi hanno mai lasciato solo nel momento del bisogno. Siete la mia seconda famiglia.

Infine voglio ringraziare Simone per avermi permesso di entrare in Digital Accademia e Luca, il mio tutor aziendale, per la disponibilità e la chiarezza con le quali ha sempre risposto ai miei dubbi. Estendo questo ringraziamento ai componenti del team di Digital Accademia e di Wethod per avermi accolto come uno di loro, dandomi la possibilità di vivere un'esperienza che difficilmente dimenticherò.

Padova, Agosto 2016 *Andrea Tombolato*

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Il contesto aziendale

1.1 H-FARM

H-FARM nasce nel 2005 con l'obiettivo di aiutare i giovani imprenditori a lanciare le loro idee innovative e supportare la trasformazione delle aziende italiane in un'ottica digitale, il tutto ospitato in un'area verde di 90.000 m^2 all'interno del parco naturale del fiume Sile.

Il 13 Novembre 2015 H-FARM viene ammessa in Borsa, all'interno del segmento AIM (Alternative Investment Market).

Oggi H-FARM è una piattaforma innovativa che:

- Sostiene la creazione di nuovi modelli d'impresa attraverso investimenti in startup$_G$;
- Guida la trasformazione delle grandi aziende verso il digitale;
- Fornisce istruzione di alto livello nelle aree del digitale a studenti e professionisti.

Il modello d'impresa di H-FARM si basa su tre pilastri fortemente interconnessi tra loro: **investimento**, **industria** ed **istruzione**.

figura 1.1: Logo H-FARM

1.1.1 Primo pilastro: investimento

La chiave per sviluppare una visione d'investimento più ampia e creare nuovi progetti innovativi è il cambiamento. Vengono quindi prese in considerazione idee provenienti da tutta Europa, selezionando quelle più promettenti ed investendo su di esse.

La strategia d'investimento prevede:

- Investimenti diretti (da 100.000 € a 500.000 €) in startup$_G$ italiane alle prime armi che abbiano ambizioni nazionali o globali nei campi SaaS e B2B$_G$;
- Investimenti strategici in *InReach Ventures*: piattaforma di investimento che permette di accedere al fiorente ecosistema delle startup$_G$ presenti in Europa. Ha sede a Londra ed è specializzata finanziamento early stage$_G$ nel settore software;
- H-CAMP: programma di accelerazione della durata di 4 mesi che mira a sostenere giovani talenti nello sviluppo dei loro modelli d'impresa.

L'obiettivo principale è quello di scoprire ed investire nelle startup$_G$ che si concentrano sulle eccellenze italiane: moda e design, cibo e vino, IoT$_G$, viaggi e turismo.

H-CAMP

H-CAMP è un programma di accelerazione internazionale e "all-inclusive", dedicato a quei talenti ed imprenditori che vogliono sviluppare le proprie idee. Ogni anno vengono lanciate due *call for ideas*: intervalli temporali entro i quali chiunque può fare domanda per partecipare ad H-CAMP.

H-CAMP fa parte del *Global Accelerator Network*, un'organizzazione internazionale che include tutti i più importanti acceleratori a livello mondiale. Questo permette alle startup$_G$ che entrano a far parte di H-CAMP di poter accedere ad una serie di agevolazioni che vanno dal mentoring$_G$ alle relazioni con investitori internazionali.

Il programma è strutturato in 3 fasi, gestite da persone responsabili con competenze tali da poter garantire i migliori risultati ad ogni fase:

1. **Selezione delle startup$_G$**: vengono scelti i progetti ritenuti più meritevoli tra quelli che hanno risposto alla call for ideas;
2. **Programma di accelerazione**: la fase di accelerazione è il cuore pulsante del progetto H-CAMP. In questa fase le startup$_G$ selezionate lavorano a stretto contatto con un team di mentor$_G$ ed esperti per sviluppare e consolidare la loro idea d'impresa. Durante questa fase, inoltre, ogni startup$_G$ riceve 20.000 € per coprire le proprie spese iniziali;
3. **Demo day**: evento conclusivo del periodo di accelerazione durante il quale le startup$_G$ accelerate presentano il proprio prodotto ad una platea di investitori, aziende, giornalisti e azionisti di H-FARM.

1.1.2 Secondo pilastro: industria

Le nuove tendenze tecnologiche dei consumatori obbligano le aziende a rivalutare le modalità con le quali investono sul personale, sui processi e sulle tecnologie. H-FARM aiuta la aziende a trasformare i loro modelli d'impresa e le loro attività in armonia

figura 1.2: Distribuzione temporale delle fasi previste dal programma H-CAMP

con i cambiamenti digitali in atto.

La strategia per l'industria propone:

- **Digital transformation**: formazione, innovazione e rinnovamento dei processi organizzativi per diventare protagonisti del cambiamento digitale in ambito aziendale;
- **H-ACK**: maratone di 24 ore focalizzate su specifici settori del"industria. I partecipanti si dividono in team e lavorano per trovare soluzioni a problemi concreti esposti dalla compagnia promotrice dell'evento.

1.1.3 Terzo pilastro: istruzione

I tradizionali sistemi educativi stanno rapidamente diventando obsoleti e la domanda di metodi di apprendimento innovativi è in rapida crescita. H-FARM è il più grande ecosistema digitale attualmente presente in Italia, che unisce le best practice del mondo accademico con nuovi metodi di apprendimento presi in prestito dal mondo del business e delle startup$_G$.

Per Maggio 2017 è prevista l'inaugurazione di una nuova area dedicata ad H-CAMPUS: il nuovo progetto formativo di H-FARM dedicato alla diffusione della cultura digitale attraverso un approccio innovativo. Studenti, professionisti e manager saranno accompagnati e resi protagonisti del processo di trasformazione digitale. H-CAMPUS nascerà all'interno di un grande spazio aperto che andrà ad ospitare una International School$_G$ (dai 6 ai 18 anni) ed una università con più di 20 tra insegnanti e formatori.

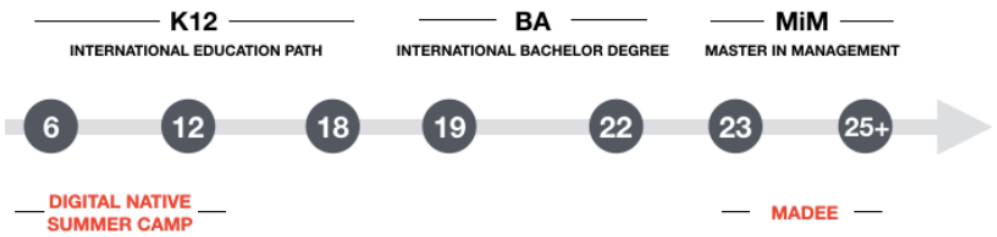


figura 1.3: Programma H-CAMPUS rivolto a diverse fasce d'età

1.2 Digital Accademia

Digital Accademia è il centro italiano di riferimento per la diffusione della cultura digitale ed è situata all'interno dell'ecosistema H-FARM.
Questo è il luogo dove studenti, professionisti e innovatori possono conoscere il potenziale del digitale, integrarlo nei propri processi di sviluppo e guardare al futuro. Il tutto viene veicolato attraverso un approccio concreto, con il quale si mira alla formazione attraverso l'esperienza.

L'offerta di Digital Accademia si rivolge principalmente alle aziende e propone strumenti di gestione della conoscenza, strumenti di comunicazione e sessioni formative che consentano di rimanere al passo con il cambiamento costante. La cultura digitale offre possibilità di interazione impensabili fino a poco tempo fa e permette di veicolare contenuti in modo nuovo, coinvolgente, personale e non convenzionale.

figura 1.4: Logo Digital Accademia

1.2.1 Formazione

Diffondere la cultura digitale in azienda significa prima di tutto venire in contatto con le persone e con le loro competenze. L'innovazione del digitale non è un concetto astratto ma una prassi, per questo non bastano corsi di aggiornamento o sessioni di formazione in aula. La formazione deve essere, prima di tutto, un'esperienza che coinvolge.

Bootcamp

All'interno dell'offerta di formazione si colloca il progetto **Bootcamp**: un'esperienza full immersion di 3 giorni e 3 notti sulla cultura digitale, ospitata in Villa Annia: il quartier generale di Digital Accademia.

Le giornate alternano momenti di teoria a sessioni di pura pratica, mentre i pranzi, le cene e le sere sono momenti sempre conviviali arricchiti dalla presenza di ospiti ed esperti del settore di pertinenza per:

- Imparare a pensare digitale;
- Capire come funziona una buona storia;
- Scoprire le logiche dietro a un e-commerce di successo;
- Costruire presentazioni persuasive.

1.2.2 Comunicazione interna

Le esperienze di comunicazione interna più efficaci parlano ai dipendenti come si parlerebbe al pubblico di un evento di *entertainment*. L'avvento massiccio del digitale ha fatto sì che non ci sia più differenza tra comunicazione esterna ed interna all'azienda.

Plot

All'interno dell'offerta per la comunicazione interna si colloca il progetto **Plot**.

Questo progetto si basa sui principi degli Alternate Reality Game$_G$ per coinvolgere e divertire le persone all'interno di una organizzazione. I dipendenti si trasformano in giocatori e partecipano alla risoluzione di enigmi come in un romanzo giallo interattivo: analizzando i file aziendali e individuando informazioni nel mondo reale devono riuscire ad arrivare alla soluzione.

Pensato inizialmente per Vodafone, Plot è un'esperienza avvincente progettata per coinvolgere gli impiegati di tutto il mondo in un gioco mirato a salvare una loro collega, Sugita, rapita da un'intelligenza artificiale da essa stessa progettata.

Sugita rivela ai partecipanti di aver lasciato alcune *backdoor* durante la fase di testing dell'intelligenza artificiale, ogni backdoor richiede una password che può essere trovata da qualche parte all'interno della intranet Vodafone. Per trovare le password, i giocatori (divisi in gruppi) devono studiare prodotti specifici dell'azienda, guardare tutorial e interagire tra loro e con i sistemi interni.

Sono previsti dieci livelli e dieci settimane di gioco, durante le quali i team provenienti da tutto il mondo si sfidano per salvare Sugita.

Terminati i dieci livelli, è prevista un'ultima fase nella quale le migliori squadre da tutto il mondo si sfidano a Londra in un'intensa giornata di gioco.

La squadra vincitrice si aggiudica le 6.000 sterline messe in palio come ricompensa.

Il Plot sviluppato per Vodafone ha registrato un grande successo, con oltre 8.000 giocatori distribuiti in 21 paesi. Questo ha portato a riproporre il format per altri clienti, tra i quali TUI Group e Assicurazioni Generali.

La struttura Perchè abbia successo, il gioco non deve essere presentato come tale ma deve confondersi quanto più possibile con la realtà degli eventi. A questo scopo, la presentazione deve avvenire all'interno del contesto lavorativo. Ad esempio, nel caso di Vodafone, il meeting annuale dei dipendenti veniva interrotto da un video nel quale l'intelligenza artificiale comunicava di aver rapito Sugita e sfidava i colleghi a salvarla.

Dopo aver effettuato la registrazione presso il sistema, la piattaforma prevede una **fase di inviti** durante la quale ogni utente può invitare un certo numero di colleghi

a far parte della propria squadra.

Al termine di questa fase, un algoritmo si occupa di formare le squadre secondo le preferenze espresse dagli utenti. Idealmente, se l'utente A sceglie l'utente B e l'utente B accetta, gli utenti A e B vengono inseriti nella stessa squadra.

Una volta chiusa la fase di inviti e formate le squadre, inizia la **fase di gioco**. La fase di gioco si articola in varie **mission**, ognuna può essere pensata come un blog che, durante la settimana, viene popolato tramite **post** appartenenti a tre possibili tipologie:

- **Testo**: introduce, in modo molto generale, l'obiettivo della mission e può avere contenuti multimediali come immagini e video;
- **Captcha**: prevede una domanda rivolta singolarmente all'utente, atta a verificare che esso sia effettivamente un umano e non una macchina;
- **Team**: prevede una domanda rivolta alla squadra.

In questa fase, ogni settimana prevede due momenti chiave:

- Ogni **lunedì** viene pubblicata una nuova mission, alla quale sono associati inizialmente un post di tipo testo e un post di tipo captcha;
- Ogni **mercoledì** viene pubblicato un post di tipo team, associato alla mission del lunedì.

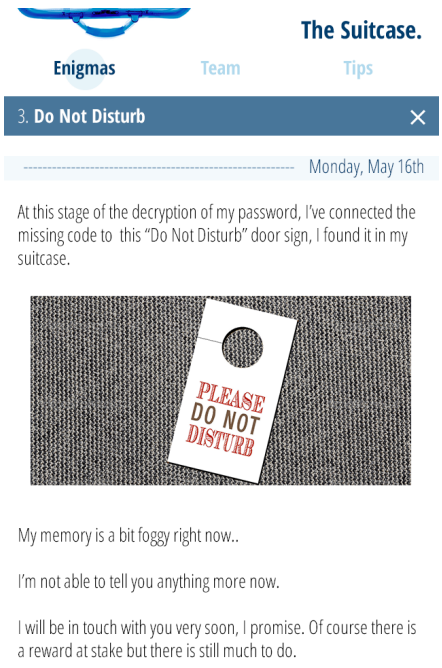


figura 1.5: Un post di tipo testo

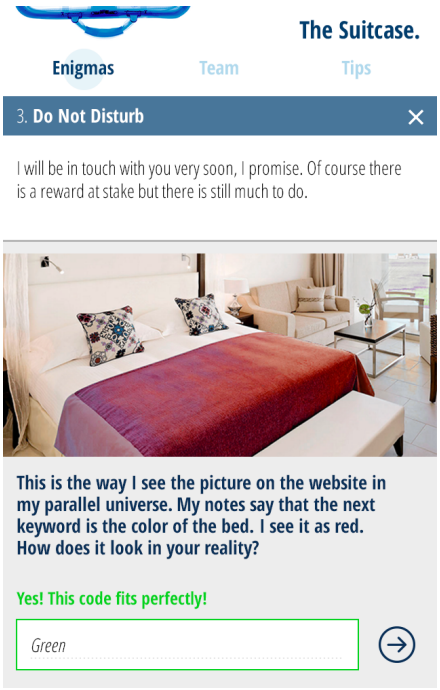


figura 1.6: Un post di tipo captcha

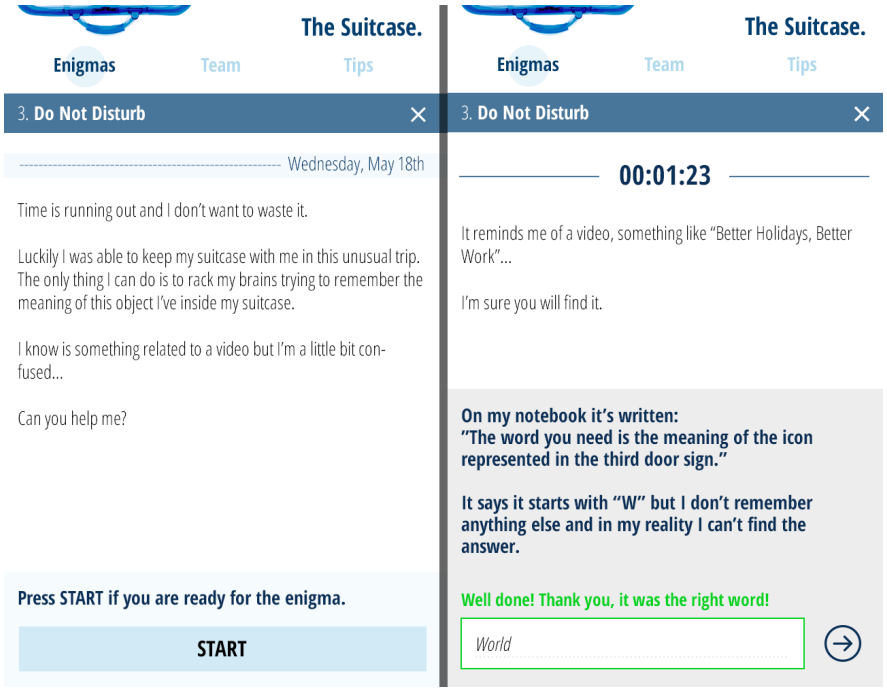


figura 1.7: Un post di tipo team

Per accedere al post di tipo team è necessario che almeno la metà dei componenti della squadra abbia risposto alla propria domanda di tipo captcha.

Le domande di tipo captcha sono uniche rispetto all'utente e alla squadra, con questo si intende che una specifica domanda di tipo captcha:

- non viene presentata allo stesso utente più di una volta;
- non viene presentata ai compagni di squadra.

La domanda associata al post di tipo team non è immediatamente visibile alla pubblicazione del post, ma può essere svelata da un componente della squadra. Una volta svelata la domanda di tipo team, viene avviato un cronometro che registra il tempo impiegato dalla squadra per rispondere.

Mentre le domande di tipo captcha sono pensate per richiedere uno sforzo minimo all'utente, quelle di tipo team hanno grado di difficoltà più alto, che aumenta col progredire del gioco.

Le domande sono pensate in modo da portare gli utenti ad affrontare specifici **touchpoint**: strumenti o concetti che l'azienda vuole far conoscere ai propri dipendenti.
Ad esempio, nel caso di Vodafone, Plot è stato progettato per perseguire i seguenti touchpoint:

- Far conoscere ai dipendenti di tutto il mondo una nuova area della intranet aziendale;
- Creare coesione tra i dipendenti;
- Attivare forme di coinvolgimento dirette attraverso la costruzione di un'esperienza;
- Trovare nuovi strumenti globali di comunicazione interna.

La squadra riceve un punteggio per ogni mission e, tale punteggio, è una funzione del tempo impiegato per rispondere alla domanda di tipo team.
Il punteggio totale di una squadra, che determina la posizione in classifica della stessa, è ricavato come somma dei punteggi ottenuti nelle varie mission.

Capitolo 2

Il progetto di stage

2.1 Il contesto

La piattaforma Plot è composta da due moduli Laravel distinti: un modulo di gioco ed un modulo CMS_G.

Il **modulo di gioco** incorpora la logica necessaria all'utente per interagire con la piattaforma. Questo modulo permette quindi agli utenti finali di:

- Autenticarsi;
- Invitare altri utenti a far parte della propria squadra;
- Accedere al contenuto delle mission;
- Accedere al contenuto dei post;
- Accedere e rispondere alle question che vengono loro proposte;
- Accedere alle informazioni riguardanti il proprio team;
- Consultare la classifica generale.

Il **modulo CMS_G** incorpora la logica necessaria agli amministratori per gestire il Plot. Questo modulo permette quindi agli amministratori di:

- Autenticarsi;
- Creare nuove mission;
- Eliminare mission esistenti;
- Modificare i dettagli di una particolare mission;
- Accedere all'elenco di tutte le mission;
- Creare nuovi post per una particolare mission;
- Eliminare post esistenti;
- Modificare i dettagli di un particolare post;

- Accedere all'elenco di tutti i post relativi ad una particolare mission;
- Creare nuove question per un particolare post;
- Eliminare question esistenti;
- Modificare i dettagli di una particolare question;
- Accedere all'elenco di tutte le question relative ad un particolare post;
- Accedere all'elenco degli utenti iscritti alla piattaforma;
- Eliminare utenti esistenti;
- Modificare i dettagli di un particolare utente;
- Accedere all'elenco dei compagni di team di un particolare utente;
- Accedere all'elenco dei team;
- Eliminare team esistenti;
- Modificare i dettagli di un particolare team;
- Accedere all'elenco dei componenti di un particolare team;
- Accedere alla classifica generale;
- Accedere all'elenco delle domande previste, corredate delle risposte corrette e del tipo di post al quale sono associate;
- Accedere all'elenco delle risposte fornite dagli utenti.

Entrambi i moduli Laravel utilizzano il medesimo database MySQL$_G$ per la persistenza dei dati (Figura 2.1). Il database, pensato inizialmente per Vodafone, è stato adattato per diversi Plot, modificandolo di volta in volta. Le continue modifiche hanno portato ad avere un database inconsistente e poco manutenibile, a causa anche della scarsa documentazione.

Ogni modulo Laravel prevede un front end$_G$ collegato ad un relativo back end$_G$. Le logiche che sottendono i due back end$_G$ sono, però, molto simili e questo porta ad avere codice duplicato e poco manutenibile.

Ulteriori problematiche nascono dal fatto che entrambi i front end$_G$ sono fortemente accoppiati al relativo back end$_G$, questo implica una struttura poco modulare e difficilmente riusabile.

Nel prosieguo della trattazione, con l'appellativo "Plot 1.0" mi riferirò alla configurazione della piattaforma esistente prima dell'inizio del progetto di stage.

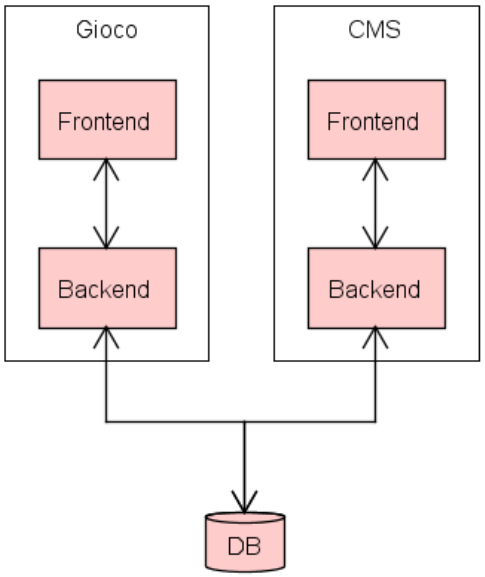


figura 2.1: Organizzazione della piattaforma prima del progetto di stage

2.2 Descrizione del progetto

Il progetto di stage prevede tre attività principali:

1. Refactoring del database che consente la persistenza dei dati utili alla piattaforma Plot;
2. Creazione di un set di API REST che permettano di interagire con il database (di cui al punto 1);
3. Refactoring del front end$_G$ relativo al CMS$_G$.

La piattaforma Plot è in continua evoluzione e mira ad adattarsi, quanto più possibile, alle richieste dei clienti. Le attività di progetto devono quindi essere affrontate con l'obiettivo di massimizzare la modularizzazione, in modo da consentire una buona manutenibilità della piattaforma ed un basso accoppiamento tra le parti.

Dato il respiro internazionale e multinazionale dei potenziali clienti della piattaforma, deve essere implementato il concetto di **traducibilità**: ogni risorsa di testo importante ai fini del gioco deve poter essere localizzata in base alla lingua preferita dall'utente che ne usufruisce.

Mantenere uno storico delle azioni effettuate dai giocatori può risultare utile per individuare la causa di eventuali problemi inaspettati, si richiede quindi di prevedere un meccanismo di **logging** che registri e renda accessibili le richieste HTTP$_G$ effettuate dai giocatori.

Il set di API REST ed il front end$_G$ del CMS$_G$ dovranno consentire di effettuare tutte le operazioni già disponibili dall'interfaccia grafica del CMS$_G$ di Plot 1.0

(elencate nella sezione "‘Il contesto"‘). In aggiunta alle operazioni già note, dovrà essere possibile:

- Accedere all'elenco degli amministratori della piattaforma;
- Eliminare gli amministratori;
- Modificare i dettagli di un particolare amministratore;
- Accedere all'elenco dei log generati dagli utenti del front end$_G$ di gioco;
- Accedere all'elenco delle traduzioni disponibili per una data risorsa;
- Eliminare le traduzioni relative ad una data risorsa;
- Modificare i dettagli di un particolare traduzione.

Non è richiesta un'interfaccia grafica professionale per il modulo CMS$_G$.

2.3 Tecnologie utilizzate

2.3.1 Database

Per gestire il database è stato scelto **MySQL Workbench**: uno strumento di amministrazione visuale che permette di creare e modificare agevolmente lo schema di un database MySQL$_G$.

La scelta è ricaduta su questo strumento in quanto:

- Ufficialmente supportato e consigliato dalla comunità MySQL;
- Fornisce un procedimento assistito per la creazione dello schema, facilitando l'individuazione di errori.

figura 2.2: Logo MySQL Workbench

2.3.2 Framework

Back end

Il framework principale utilizzato per il progetto è **Laravel**: un framework PHP open-source, pensato per la creazione di applicazioni web basate sul pattern architetturale MVC (Model-View-Controller).

Il framework è stato scelto dall'azienda per i vantaggi che offre, tra i quali:

- Una documentazione ampia e facilmente consultabile in rete;
- Una sintassi pulita ed intuitiva;
- Facilità di implementazione della dependency injection;
- *Eloquent*: ORM$_G$ integrato di Laravel;
- *Blade*: il template engine integrato di Laravel.

figura 2.3: Logo Laravel

Front end

Per il refactoring dell'interfaccia grafica del CMS$_G$ è stato utilizzato **Bootstrap**: un framework HTML, CSS e JavaScript per la creazione di pagine web responsive e mobile-first.

Il framework è stato scelto perchè:

- Mette a disposizione tutti i componenti base e gli stili utili per la creazione di una pagina web (layout a griglia, tabelle, box modali, form e molto altro);
- Viene supportato da tutti i maggiori browser;
- Permette di creare interfacce grafiche consistenti, anche senza modificare le componenti fornite;
- Pensato per creare pagine responsive che si adattino anche ai dispositivi mobile;
- Include un set di icone pronte all'uso;
- Ha una buona documentazione ed un grande supporto dalla comunità;
- Esistono molti temi, template e plugin gratuiti.

Oltre ai molti pregi, questo framework porta con sé almeno due difetti:

- Produce codice HTML verboso che può risultare non perfettamente semantico;;
- Molti componenti non funzionano se JavaScript è disattivato.

In accordo con l'azienda, questi non sono stati ritenuti problemi bloccanti in quanto il CMS$_G$ verrà ospitato su un server privato e sarà ad uso esclusivo degli amministratori, che avranno cura di utilizzare un browser che supporti JavaScript.

figura 2.4: Logo Bootstrap

2.3.3 Versionamento

Per il versionamento del codice è stato è stato utilizzato **Bitbucket** con base **Git**, in quanto già in uso presso l'azienda. Bitbucket è un servizio di web-hosting per progetti software che utilizzano Git o Mercurial come sistemi di versionamento.

Il punto di forza di Bitbucket è la possibilità di creare, gratuitamente, repository privati. Questo risulta fondamentale per un'azienda che voglia mantenere il codice dei propri progetti accessibile solo a determinate persone.

figura 2.5: Logo Bitbucket

Git è un software per il versionamento distribuito ed è stato scelto dall'azienda in quanto offre:

- Semplicità di utilizzo;
- Una buona documentazione disponibile online;
- La possibilità di lavorare anche offline, in quanto ogni sviluppatore possiede una copia locale del repository sul quale può effettuare commit anche in assenza di connessione.

figura 2.6: Logo Git

2.3.4 Gestione di progetto

Per la gestione di progetto è stato è stato utilizzato **Basecamp**, in quanto già in uso presso l'azienda.

Basecamp è un servizio web per che permette di condividere e organizzare tutto ciò che è necessario per la gestione di un progetto: task, discussioni, deadline e file.

figura 2.7: Logo Basecamp

Capitolo 3

Analisi dei requisiti

Sin dalla sua concezione, la piattaforma Plot è stata pensata come un prodotto unico e fatto su misura per il cliente. Questo ha impedito la definizione di una struttura di base sulla quale innestare eventuali cambiamenti.
La prima attività del progetto è stata dunque la consultazione dello schema del database e del codice relativo al CMS_G di Plot 1.0. Successivamente sono state intervistate le persone coinvolte nello sviluppo di Plot 1.0: developer, responsabili dei contenuti ed utilizzatori del CMS_G in modo da capire cosa non li soddisfacesse nella struttura attuale e cosa potesse essere fatto per migliorarla. Da queste interviste sono stati sintetizzati i requisiti che il database ed il CMS_G avrebbero dovuto avere. Tali requisiti sono stati fondamentali per la successiva progettazione ma possono risultare utili anche all'azienda per valutare in modo più razionale limiti e possibilità della piattaforma.

3.1 Requisiti del database

3.1.1 Descrizione dei requisiti

Un **consumer** è un concetto astratto che rappresenta un generico utilizzatore della piattaforma Plot, esso è caratterizzato da nome, cognome, email e password. Un consumer può concretizzarsi nel concetto di admin o user.
Un **admin** rappresenta un amministratore del CMS_G, ha quindi l'autorizzazione di accedere al CMS_G ma non può accedere alla parte di gioco.
Uno **user** rappresenta un utente del gioco e può quindi accedere alla parte di gioco ma non al CMS_G.
Solitamente i consumer si autenticano presso la piattaforma tramite email e password ma può capitare che gli user utilizzino un servizio di autenticazione esterno tramite *token*.

Un **invitation** rappresenta un invito che può essere spedito o ricevuto da uno user durante la fase di inviti. Un invitation è caratterizzato dal mittente e dal destinatario, che dovranno essere entrambi user.

Terminata la fase di inviti, ogni user appartiene ad un **team** caratterizzato da un nome e da un'immagine rappresentativa dello stesso. I team vengono creati a

priori e sono associati agli user solo al termine della fase di inviti.

Qualsiasi richiesta HTTP$_G$ effettuata da uno user nel front end di gioco genera un **log** contenente:

- Un riferimento allo user che ha effettuato la richiesta;
- Indirizzo IP del dispositivo che ha effettuato la richiesta;
- Il metodo HTTP$_G$ utilizzato per la richiesta;
- L'URL$_G$ al quale è stata inoltrata la richiesta;
- Lo user agent che ha effettuato la richiesta;
- Eventuali dati aggiuntivi trasportati dalla richiesta.

Un **language** rappresenta una lingua parlata ed è caratterizzato da un nome (es. Italiano) e da un codice identificativo (es. it).
Una **translation** rappresenta una traduzione disponibile per un certo contenuto in un particolare language. Ogni user può indicare il proprio language preferito e questa informazione può essere utilizzata per fornire i contenuti di gioco localizzati in base allo user.
Ogni contenuto traducibile deve prevedere un testo di default da utilizzare nel caso in cui non siano state previste traduzioni per la lingua scelta dallo user.

Uno user può appartenere ad una **category**, il significato della category non è fissato a priori ma può essere istanziato in base alle necessità del progetto e del cliente che lo richiede. Il concetto di category è quindi astratto e viene utilizzato per partizionare gli user sulla base di una o più caratteristiche. Ad esempio, gli user possono essere categorizzati per regione di appartenenza o per posizione lavorativa. Una category ha un nome che la identifica.
Per questo progetto vengono istanziate le due category più usate nei Plot passati: region e business role.
Una **region** rappresenta l'area del mondo alla quale appartiene un certo user ed è caratterizzata da una timezone. Una **timezone** identifica un particolare fuso orario ed è caratterizzata dal nome (es. Central European Time), dal codice identificativo della timezone (es. CET) e dall'offset rispetto allo UTC (es. +1). Una region può quindi rappresentare un insieme di stati, un singolo stato o una porzione di esso, questo dipende dallo specifica istanza di Plot.
Un **business role** rappresenta il ruolo dello user all'interno dell'azienda o, più in generale, l'area lavorativa alla quale esso afferisce.

Un Plot è composto da una o più **mission**. Ogni mission è caratterizzata da un nome, un'immagine rappresentativa, una data d'inizio e una data di fine, oltre la quale la mission si considera chiusa. Una mission è composta da più **post**. Ogni post prevede un testo ed appartiene ad una delle seguenti tipologie:

- **Text post**;
- **Captcha post**: prevede una domanda indirizzata ai singoli user;

- **Team post**: prevede una domanda indirizzata ai team.

Una **question** è un concetto astratto che rappresenta una generica domanda, essa prevede un testo che riporta il corpo della domanda stessa. Una question può essere istanziata da una user question o da una team question.
Una **user question** rappresenta una domanda indirizzata ai singoli user. Le user question sono caratterizzate dalle category alle quali sono rivolte. Se il concetto di category è utilizzato dalla particolare istanza di Plot, è possibile assegnare domande diverse a user appartenenti a diverse category. Viceversa, se il concetto di category non è utilizzato, ogni domanda può potenzialmente essere assegnata ad ogni user. Una user question è visibile alla pubblicazione del captcha post al quale appartiene.
Una **team question** rappresenta una domanda indirizzata ai team. Le team question prevedono un tempo limite entro il quale devono ricevere risposta. Qualsiasi componente del team può fornire una o più risposte ad una specifica domanda. Una team question non è immediatamente visibile alla pubblicazione del team post al quale appartiene, per svelarla è necessario che un componente del team dichiari di volerla visualizzare, a quel punto la domanda è visualizzabile da tutti i membri del team. Nel momento in cui uno user decide di svelare una domanda, viene inizializzato un timer che decreta il tempo impiegato dal team per fornire una risposta corretta. Una funzione del tempo impiegato andrà a decretare il punteggio totalizzato dal team nella specifica mission. Il punteggio totale di un team è dato dalla somma dei punteggi totalizzati dal team nelle varie mission e, sulla base di questo, viene stilata la classifica dei team.

Una **correct answer** rappresenta una risposta corretta prevista per una specifica question, ogni question deve prevedere almeno una risposta corretta. Ogni user può fornire più risposte alle question proposte, una risposta ad una question è considerata corretta se e solo se il suo contenuto compare tra le correct answer previste per la specifica question.

Uno user può visualizzare un post più volte e, lo stesso post, può essere visualizzato da user diversi.

Un admin può decidere se mostrare (o mantenere nascosta) agli user una mission o un post.

Deve essere possibile fornire una translation per i contenuti testuali di mission, post, question ed in generale tutti gli elementi di gioco con i quali entra in contatto lo user.

3.1.2 L'integrazione con Eloquent

Per poter sfruttare appieno le potenzialità di *Eloquent*, l'ORM fornito da Laravel, è necessario rispettare alcune convenzioni:

- Lo schema di ogni tabella appartenente al database deve prevedere le colonne `created_at` e `updated_at`. Il valore relativo a queste colonne viene automaticamente inserito dall'ORM ogniqualvolta venga creato o modificato un record;

- Lo schema di ogni tabella appartenente al database deve prevedere un singolo attributo che funga da chiave primaria. Tale attributo deve essere chiamato `id`;
- I nomi delle tabelle e degli attributi devono essere in minuscolo e seguire la convenzione snake case$_G$;
- I nomi delle tabelle devono essere pensati al plurale. Ad esempio, la tabella contenente i captcha post dovrà essere chiamata `captcha_posts`;
- Gli attributi che fungono da chiave esterna devono essere nominati posponendo "id" al nome singolare della tabella alla quale si riferiscono. Ad esempio, una chiave esterna verso la tabella users dovrà essere chiamata `user_id`.

3.2 Requisiti del CMS

La piattaforma si rivolge principalmente a due categorie di utenti:

- L'**utente web**: accede alle funzionalità offerte dalla piattaforma attraverso un browser web e la relativa interfaccia grafica;
- L'**utente API**: accede alle funzionalità offerte dalla piattaforma attraverso le API REST da essa esposte.

3.2.1 Caso d'uso UCG: caso d'uso generale

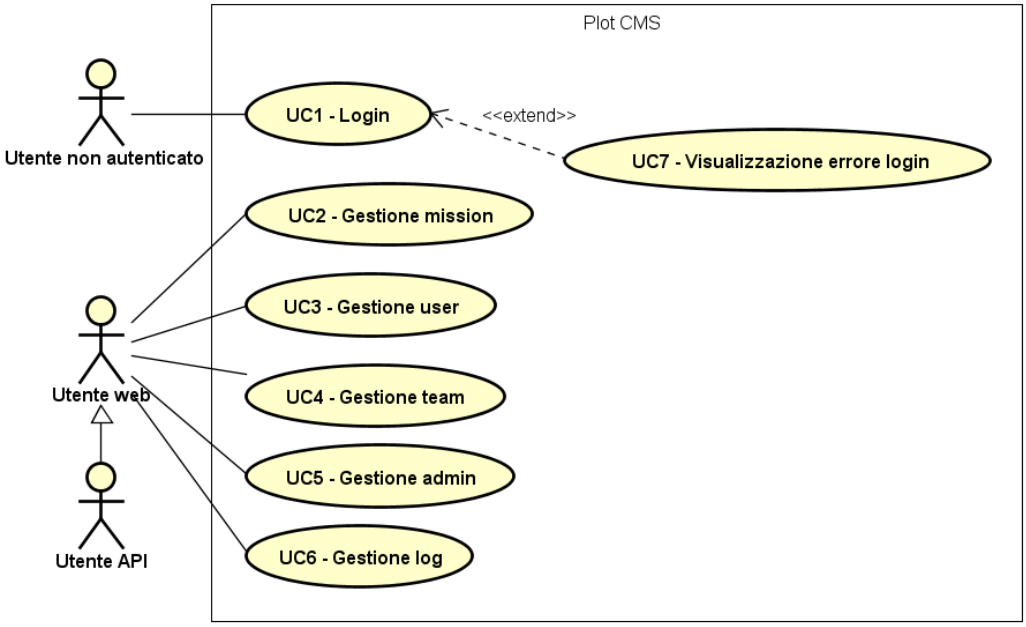


figura 3.1: Caso d'uso UCG: caso d'uso generale

- **Attori**: utente non autenticato, utente web, utente API;

- **Descrizione**: un utente deve poter, innanzitutto, effettuare l'autenticazione presso la piattaforma per poter accedere alle funzionalità offerte. Una volta effettuata l'autenticazione, l'utente deve poter gestire: mission, amministratori della piattaforma, utenti del gioco, team del gioco e log generati dagli utenti del gioco;

- **Precondizione**: l'utente accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema;

- **Flusso principale degli eventi**:

  1. L'utente può autenticarsi presso la piattaforma (UC1);
  2. L'utente autenticato può gestire le mission previste dal Plot (UC2);
  3. L'utente autenticato può gestire gli utenti del gioco;
  4. L'utente autenticato può gestire i team del gioco;
  5. L'utente autenticato può gestire gli amministratori della piattaforma;
  6. L'utente autenticato può gestire i log generati dagli utenti del gioco.

- **Postcondizione**: il sistema ha erogato le funzionalità richieste dall'utente;

- **Scenari alternativi**:

  – Nel caso in cui l'utente provi ad autenticarsi utilizzando una combinazione di username e password non riconosciuta dal sistema:

    1. Viene presentato un errore esplicativo (UC7).

3.2.2 Caso d'uso UC1: login

- **Attori**: utente web;

- **Descrizione**: gli amministratori del Plot devono poter autenticarsi presso la piattaforma in modo da poter accedere alle funzionalità offerte;

- **Precondizione**: l'utente non è già autenticato ed accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema;

- **Flusso principale degli eventi**:

  1. L'utente può inserire una email;
  2. L'utente può inserire una password.

- **Postcondizione**: l'utente viene riconosciuto dal sistema e può accedere alle funzionalità offerte.

3.2.3 Caso d'uso UC2: gestione mission

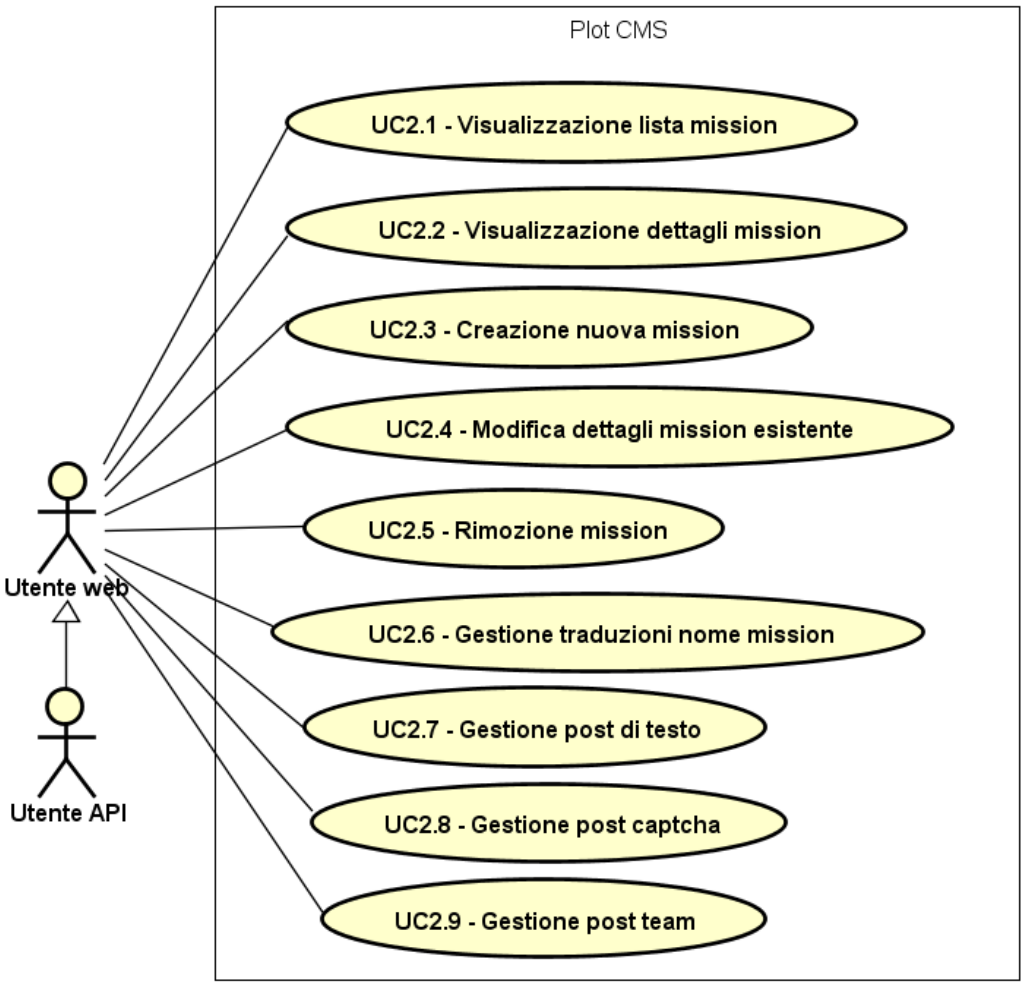


figura 3.2: Caso d'uso UC2: gestione mission

- **Attori**: utente web, utente API;
- **Descrizione**: un utente autenticato deve poter gestire gli aspetti di base legati alle mission: creazione, visualizzazione, modifica e rimozione. Inoltre deve essere possibile gestire i post legati ad una specifica mission e le traduzioni relative al nome di una specifica mission;
- **Precondizione**: l'utente accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema;
- **Flusso principale degli eventi**:
  1. L'utente autenticato può visualizzare la lista delle mission;
  2. L'utente autenticato può visualizzare i dettagli di una specifica mission (UC2.2);

3. L'utente autenticato può creare una nuova mission (UC2.3);
4. L'utente autenticato può modificare i dettagli di una specifica mission (UC2.4);
5. L'utente autenticato può rimuovere una mission;
6. L'utente autenticato può gestire le traduzioni relative al nome di una specifica mission (UC2.6);
7. L'utente autenticato può gestire i post di testo collegati ad una specifica mission (UC2.7);
8. L'utente autenticato può gestire i post di tipo captcha collegati ad una specifica mission (UC2.8);
9. L'utente autenticato può gestire i post di tipo team collegati ad una specifica mission (UC2.9).

- **Postcondizione**: il sistema ha erogato le funzionalità richieste dall'utente.

3.2.4 Caso d'uso UC2.2: visualizzazione dettagli mission

- **Attori**: utente web, utente API;
- **Descrizione**: l'utente autenticato deve poter visualizzare i dettagli relativi ad una specifica mission;
- **Precondizione**: l'utente accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema. La mission che si vuole visualizzare deve esistere;
- **Flusso principale degli eventi**:
  1. L'utente autenticato visualizza il nome di default della mission;
  2. L'utente autenticato visualizza l'URL_G dell'immagine rappresentativa della mission;
  3. L'utente autenticato visualizza la data d'inizio della mission;
  4. L'utente autenticato visualizza la data di fine della mission;
  5. L'utente autenticato visualizza il grado di visibilità della mission. Questo indica se la mission è visibile o meno agli utenti di gioco.
- **Postcondizione**: il sistema mostra all'utente i dettagli della mission richiesta.

3.2.5 Caso d'uso UC2.3: creazione nuova mission

- **Attori**: utente web, utente API;
- **Descrizione**: l'utente autenticato deve poter creare una nuova mission;
- **Precondizione**: l'utente accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema;
- **Flusso principale degli eventi**:

1. L'utente autenticato inserisce il nome di default della mission;
2. L'utente autenticato può inserire l'URL$_G$ di un'immagine rappresentativa della mission;
3. L'utente autenticato può inserire la data di inizio della mission;
4. L'utente autenticato può inserire la data di fine della mission;
5. L'utente autenticato può inserire il grado di visibilità della mission. Questo indica se la mission è visibile o meno agli utenti di gioco.

- **Postcondizione**: il sistema crea una nuova mission e ne mostra i dettagli all'utente.

3.2.6 Caso d'uso UC2.4: modifica dettagli mission esistente

- **Attori**: utente web, utente API;
- **Descrizione**: l'utente autenticato deve poter modificare i dettagli relativi ad una mission esistente;
- **Precondizione**: l'utente accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema. La mission che si vuole modificare deve esistere;
- **Flusso principale degli eventi**:
  1. L'utente autenticato può modificare il nome di default della mission;
  2. L'utente autenticato può modificare il l'URL$_G$ dell'immagine rappresentativa della mission;
  3. L'utente autenticato può modificare la data d'inizio della mission;
  4. L'utente autenticato può modificare la data di fine della mission;
  5. L'utente autenticato può modificare il grado di visibilità della mission. Questo indica se la mission è visibile o meno agli utenti di gioco.
- **Postcondizione**: il sistema apporta le modifiche richieste e mostra i dettagli della mission modificata all'utente.

3.2.7 Caso d'uso UC2.6: gestione traduzioni nome mission

- **Attori**: utente web, utente API;
- **Descrizione**: un utente autenticato deve poter gestire gli aspetti di base legati alle traduzioni del nome di una mission: creazione, visualizzazione, modifica e rimozione;
- **Precondizione**: l'utente accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema;
- **Flusso principale degli eventi**:

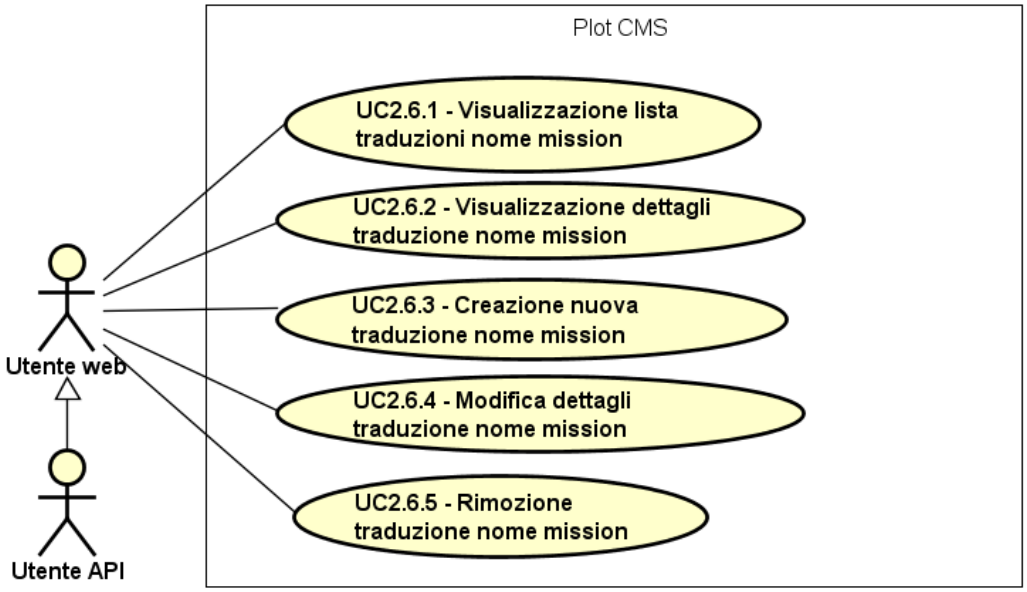


figura 3.3: Caso d'uso UC2.6: gestione traduzioni nome mission

1. L'utente autenticato può visualizzare la lista delle traduzioni previste per il nome della mission;
2. L'utente autenticato può visualizzare i dettagli di una particolare traduzione prevista per il nome della mission (UC2.6.2);
3. L'utente autenticato può creare una nuova traduzione per il nome della mission (UC2.6.3);
4. L'utente autenticato può modificare i dettagli di una particolare traduzione per il nome della mission (UC2.6.4);
5. L'utente autenticato può rimuovere una traduzione relativa al nome della mission.

- **Postcondizione**: il sistema ha erogato le funzionalità richieste dall'utente.

3.2.8 Caso d'uso UC2.6.2: visualizzazione dettagli traduzione nome mission

- **Attori**: utente web, utente API;
- **Descrizione**: l'utente autenticato deve poter visualizzare i dettagli relativi ad una specifica traduzione relativa al nome di una mission;
- **Precondizione**: l'utente accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema. La traduzione che si vuole visualizzare deve esistere;
- **Flusso principale degli eventi**:

1. L'utente autenticato visualizza la traduzione del nome della mission;
2. L'utente autenticato visualizza la lingua relativa alla traduzione .

- **Postcondizione**: il sistema mostra all'utente i dettagli dell'a traduzione richiesta.

3.2.9 Caso d'uso UC2.6.3: creazione nuova traduzione nome mission

- **Attori**: utente web, utente API;
- **Descrizione**: l'utente autenticato deve poter creare una nuova traduzione relativa al nome della mission. La nuova traduzione è caratterizzata dalla lingua, in quanto non possono esistere due traduzioni fornite nella medesima lingua e riferite al medesimo contenuto;
- **Precondizione**: l'utente accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema;
- **Flusso principale degli eventi**:
  1. L'utente inserisce il testo della traduzione relativa al nome della mission;
  2. L'utente inserisce la lingua della traduzione relativa al nome della mission.
- **Postcondizione**: il sistema crea una nuova traduzione e ne mostra i dettagli all'utente.

3.2.10 Caso d'uso UC2.6.4: modifica dettagli traduzione nome mission

- **Attori**: utente web, utente API;
- **Descrizione**: l'utente autenticato deve poter modificare i dettagli relativi ad una traduzione esistente;
- **Precondizione**: l'utente accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema. La traduzione che si vuole modificare deve esistere;
- **Flusso principale degli eventi**:
  1. L'utente autenticato può modificare il testo della traduzione relativa al nome della mission;
  2. L'utente autenticato può modificare la lingua della traduzione relativa al nome della mission.
- **Postcondizione**: il sistema apporta le modifiche richieste e mostra i dettagli della traduzione modificata all'utente.

3.2.11 Caso d'uso UC2.7: gestione post di testo

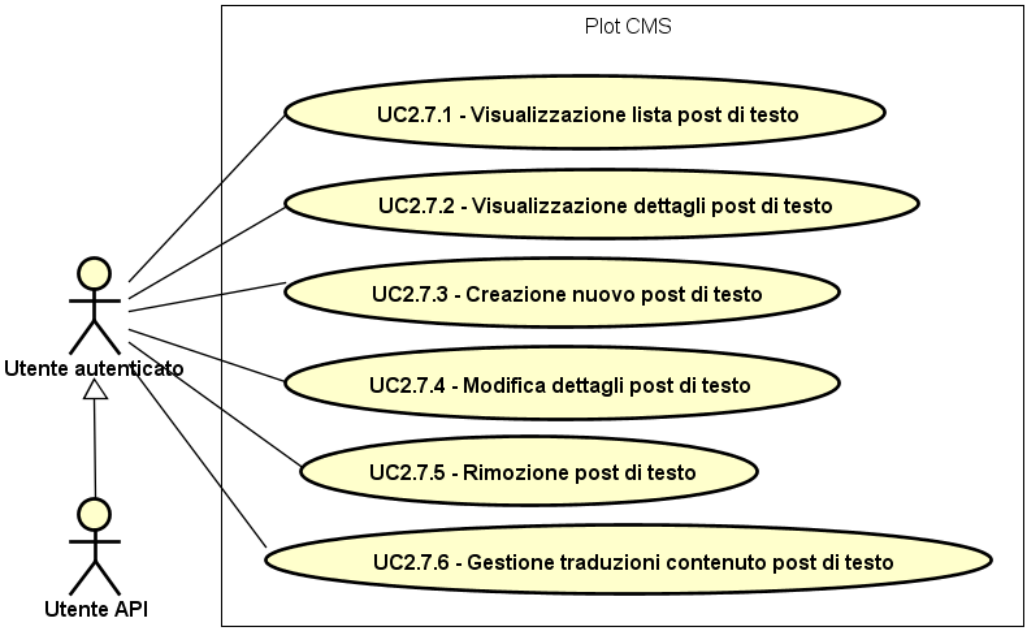


figura 3.4: Caso d'uso UC2.7: gestione post di testo

- **Attori**: utente web, utente API;
- **Descrizione**: un utente autenticato deve poter gestire gli aspetti di base legati ai post di testo: creazione, visualizzazione, modifica e rimozione. Inoltre deve essere possibile gestire le traduzioni relative al contenuto di uno specifico post di testo. Ogni post di testo è collegato ad una specifica mission;
- **Precondizione**: l'utente accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema. I post di testo hanno significato solo come figli di una mission, tale mission deve quindi esistere per poter accedere alle funzionalità di gestione dei post di testo;
- **Flusso principale degli eventi**:
  1. L'utente autenticato può visualizzare una lista di tutti i post di testo relativi alla mission;
  2. L'utente autenticato può visualizzare i dettagli di uno specifico post di testo collegato alla mission (UC2.7.2);
  3. L'utente autenticato può creare un nuovo post di testo collegato alla mission (UC2.7.3);
  4. L'utente autenticato può modificare i dettagli di un particolare post di testo collegato alla mission (UC2.7.4);
  5. L'utente autenticato può rimuovere un particolare post di testo collegato alla mission (UC2.7.5);

6. L'utente autenticato può gestire le traduzioni del contenuto di un particolare post di testo collegato alla mission (UC2.7.6).

- **Postcondizione**: il sistema ha erogato le funzionalità richieste dall'utente.

3.2.12 Caso d'uso UC2.7.2: visualizzazione dettagli post di testo

- **Attori**: utente web, utente API;
- **Descrizione**: l'utente autenticato deve poter visualizzare i dettagli relativi ad uno specifico post di testo collegato alla mission;
- **Precondizione**: l'utente autenticato accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema. Il post di testo che si vuole visualizzare deve esistere;
- **Flusso principale degli eventi**:
  1. L'utente autenticato visualizza il contenuto di default del post di testo;
  2. L'utente autenticato visualizza il grado di visibilità del post di testo. Questo indica se il post è visibile o meno agli utenti di gioco.
- **Postcondizione**: il sistema mostra all'utente i dettagli del post di testo richiesto.

3.2.13 Caso d'uso UC2.7.3: creazione nuovo post di testo

- **Attori**: utente web, utente API;
- **Descrizione**: l'utente autenticato deve poter creare un nuovo post di testo, collegandolo alla mission;
- **Precondizione**: l'utente accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema;
- **Flusso principale degli eventi**:
  1. L'utente autenticato inserisce il contenuto di default del post di testo;
  2. L'utente autenticato inserisce il grado di visibilità del post di testo. Questo indica se il post è visibile o meno agli utenti di gioco.
- **Postcondizione**: il sistema crea una nuovo post di testo, collegato alla mission, e ne mostra i dettagli all'utente.

3.2.14 Caso d'uso UC2.7.4: modifica dettagli post di testo

- **Attori**: utente web, utente API;
- **Descrizione**: l'utente autenticato deve poter modificare i dettagli relativi ad un post di testo esistente;

- **Precondizione**: l'utente accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema. Il post di testo che si vuole modificare deve esistere;
- **Flusso principale degli eventi**:
  1. L'utente autenticato può modificare il contenuto di default del post di testo;
  2. L'utente autenticato può modificare il grado di visibilità del post di testo. Questo indica se il post è visibile o meno agli utenti di gioco.
- **Postcondizione**: il sistema apporta le modifiche richieste e mostra i dettagli del post di testo modificato all'utente.

3.2.15 Caso d'uso UC2.7.5: rimozione post di testo

- **Attori**: utente web, utente API;
- **Descrizione**: l'utente autenticato deve poter rimuovere un post di testo;
- **Precondizione**: l'utente autenticato accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema. Il post di testo che si vuole rimuovere deve esistere;
- **Flusso principale degli eventi**:
  1. L'utente autenticato rimuove un particolare post di testo.
- **Postcondizione**: Il post di testo viene rimosso dal sistema e non può più essere recuperato. Il sistema mostra all'utente una lista dei post di testo collegati alla mission e ancora presenti nel sistema.

3.2.16 Caso d'uso UC2.7.6: gestione traduzioni contenuto post di testo

- **Attori**: utente web, utente API;
- **Descrizione**: un utente autenticato deve poter gestire gli aspetti di base legati alle traduzioni del contenuto di un post di testo: creazione, visualizzazione, modifica e rimozione;
- **Precondizione**: l'utente accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema;
- **Flusso principale degli eventi**:
  1. L'utente autenticato può visualizzare la lista delle traduzioni previste per il contenuto del post di testo;
  2. L'utente autenticato può visualizzare i dettagli di una particolare traduzione prevista per il contenuto del post di testo;

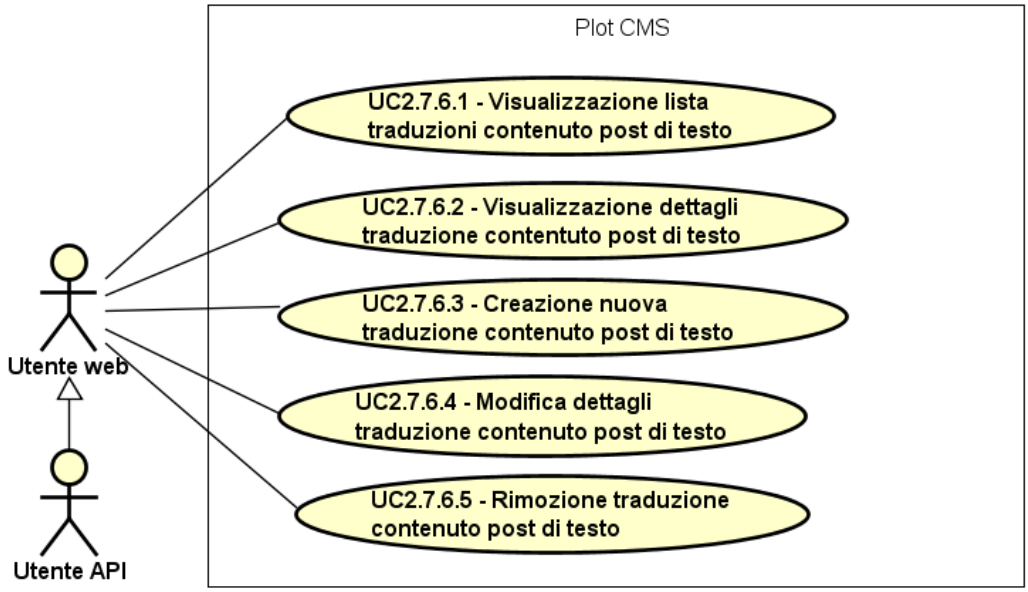


figura 3.5: Caso d'uso UC2.7.6: gestione traduzioni contenuto post di testo

3. L'utente autenticato può creare una nuova traduzione per il contenuto del post di testo;
4. L'utente autenticato può modificare i dettagli di una particolare traduzione per il contenuto del post di testo;
5. L'utente autenticato può rimuovere una traduzione relativa al contenuto del post di testo.

- **Postcondizione**: il sistema ha erogato le funzionalità richieste dall'utente.

3.2.17 Caso d'uso UC2.8: gestione post captcha

- **Attori**: utente web, utente API;
- **Descrizione**: un utente autenticato deve poter gestire gli aspetti di base legati ai post captcha: creazione, visualizzazione, modifica e rimozione. Inoltre deve essere possibile gestire le domande legate ad uno specifico post captcha e le traduzioni relative al contenuto di uno specifico post captcha;
- **Precondizione**: l'utente accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema;
- **Flusso principale degli eventi**:
  1. L'utente autenticato può visualizzare una lista di tutti i post captcha relativi alla mission;
  2. L'utente autenticato può visualizzare i dettagli di uno specifico post captcha collegato alla mission (UC2.8.2);

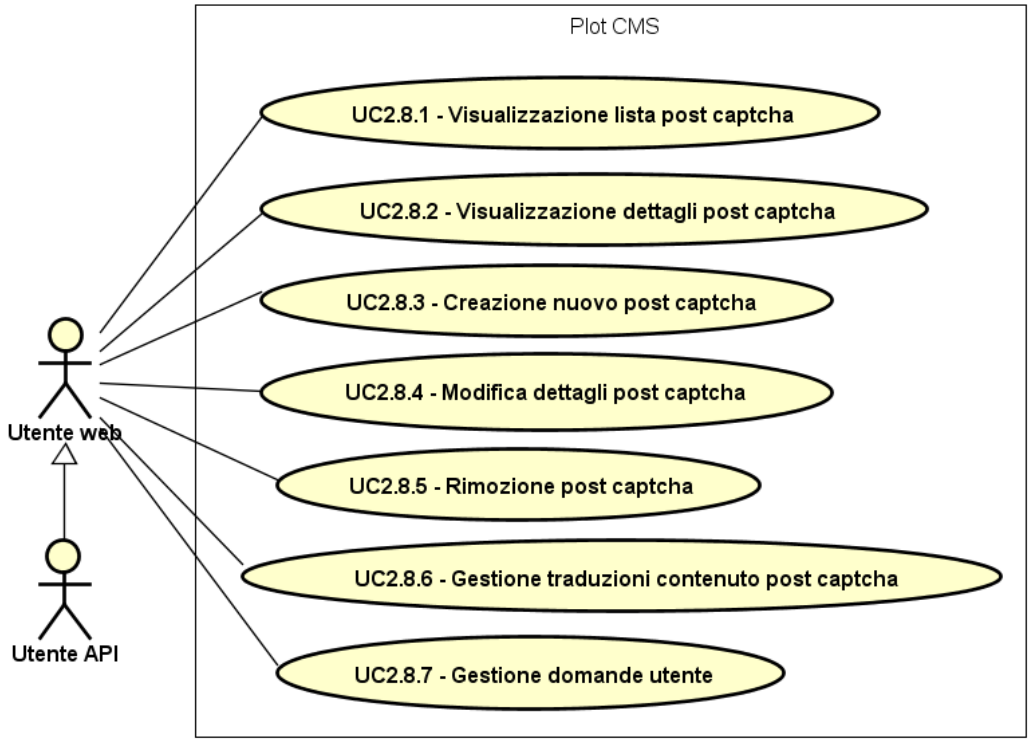


figura 3.6: Caso d'uso UC2.8: gestione post captcha

3. L'utente autenticato può creare un nuovo post captcha collegato alla mission (UC2.8.3);
4. L'utente autenticato può modificare i dettagli di un particolare post captcha collegato alla mission (UC2.8.4);
5. L'utente autenticato può rimuovere un particolare post captcha collegato alla mission (UC2.8.5);
6. L'utente autenticato può gestire le traduzioni del contenuto di un particolare post captcha collegato alla mission (UC2.8.6);
7. L'utente autenticato può gestire le domande collegate al post captcha (UC2.8.7).

- **Postcondizione**: il sistema ha erogato le funzionalità richieste dall'utente.

3.2.18 Caso d'uso UC2.8.2: visualizzazione dettagli post captcha

- **Attori**: utente web, utente API;
- **Descrizione**: l'utente autenticato deve poter visualizzare i dettagli relativi ad uno specifico post captcha collegato alla mission;
- **Precondizione**: l'utente autenticato accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema. Il post captcha che si vuole visualizzare deve esistere;

- **Flusso principale degli eventi**:
  1. L'utente autenticato visualizza il contenuto di default del post captcha;
  2. L'utente autenticato visualizza il grado di visibilità del post captcha. Questo indica se il post è visibile o meno agli utenti di gioco.
- **Postcondizione**: il sistema mostra all'utente i dettagli del post captcha richiesto.

3.2.19 Caso d'uso UC2.8.3: creazione nuovo post captcha

- **Attori**: utente web, utente API;
- **Descrizione**: l'utente autenticato deve poter creare un nuovo post captcha, collegandolo alla mission;
- **Precondizione**: l'utente accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema;
- **Flusso principale degli eventi**:
  1. L'utente autenticato inserisce il contenuto di default del post captcha;
  2. L'utente autenticato può inserire il grado di visibilità del post captcha. Questo indica se il post è visibile o meno agli utenti di gioco.
- **Postcondizione**: il sistema crea una nuovo post captcha, collegato alla mission, e ne mostra i dettagli all'utente.

3.2.20 Caso d'uso UC2.8.4: modifica dettagli post captcha

- **Attori**: utente web, utente API;
- **Descrizione**: l'utente autenticato deve poter modificare i dettagli relativi ad un post captcha esistente;
- **Precondizione**: l'utente accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema. Il post captcha che si vuole modificare deve esistere;
- **Flusso principale degli eventi**:
  1. L'utente autenticato può modificare il contenuto di default del post captcha;
  2. L'utente autenticato può modificare il grado di visibilità del post captcha. Questo indica se il post è visibile o meno agli utenti di gioco;
  3. L'utente autenticato può modificare il grado di visibilità del post captcha. Questo indica se il post è visibile o meno agli utenti di gioco;
  4. L'utente autenticato può modificare il grado di visibilità del post captcha. Questo indica se il post è visibile o meno agli utenti di gioco.
- **Postcondizione**: il sistema apporta le modifiche richieste e mostra i dettagli del post captcha modificato all'utente.

3.2.21 Caso d'uso UC2.8.5: rimozione post captcha

- **Attori**: utente web, utente API;
- **Descrizione**: l'utente autenticato deve poter rimuovere un post captcha;
- **Precondizione**: l'utente autenticato accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema. Il post captcha che si vuole rimuovere deve esistere;
- **Flusso principale degli eventi**:
  1. L'utente autenticato rimuove un particolare post captcha.
- **Postcondizione**: Il post captcha viene rimosso dal sistema e non può più essere recuperato. Il sistema mostra all'utente una lista dei post captcha collegati alla mission e ancora presenti nel sistema.

3.2.22 Caso d'uso UC2.8.6: gestione traduzioni contenuto post captcha

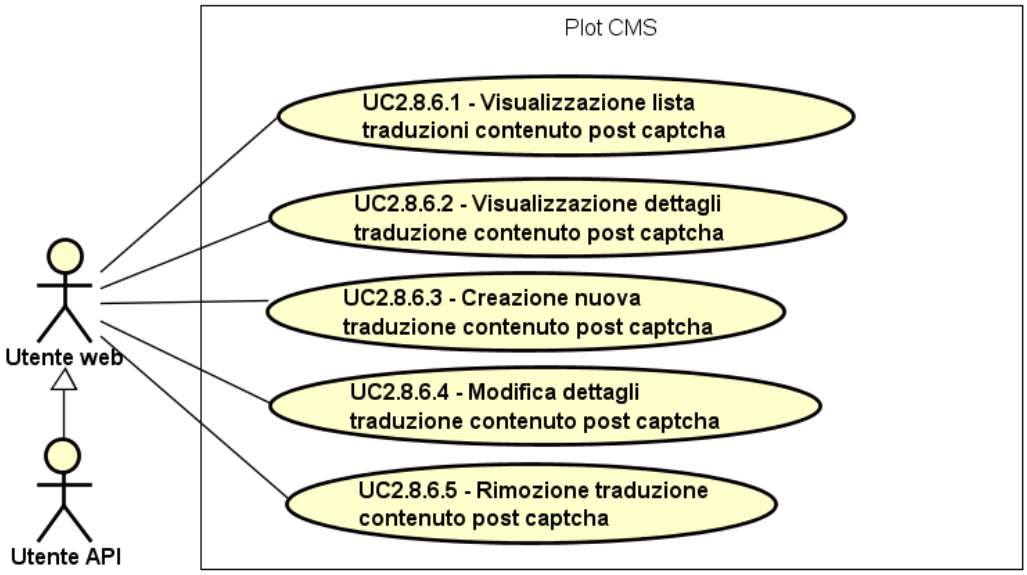


figura 3.7: Caso d'uso UC2.8.6: gestione traduzioni contenuto post captcha

- **Attori**: utente web, utente API;
- **Descrizione**: un utente autenticato deve poter gestire gli aspetti di base legati alle traduzioni del contenuto di un post captcha: creazione, visualizzazione, modifica e rimozione;
- **Precondizione**: l'utente accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema;

- **Flusso principale degli eventi**:

  1. L'utente autenticato può visualizzare la lista delle traduzioni previste per il contenuto del post captcha;
  2. L'utente autenticato può visualizzare i dettagli di una particolare traduzione prevista per il contenuto del post captcha;
  3. L'utente autenticato può creare una nuova traduzione per il contenuto del post captcha;
  4. L'utente autenticato può modificare i dettagli di una particolare traduzione per il contenuto del post captcha;
  5. L'utente autenticato può rimuovere una traduzione relativa al contenuto del post captcha.

- **Postcondizione**: il sistema ha erogato le funzionalità richieste dall'utente.

3.2.23 Caso d'uso UC2.8.7: gestione domande utente

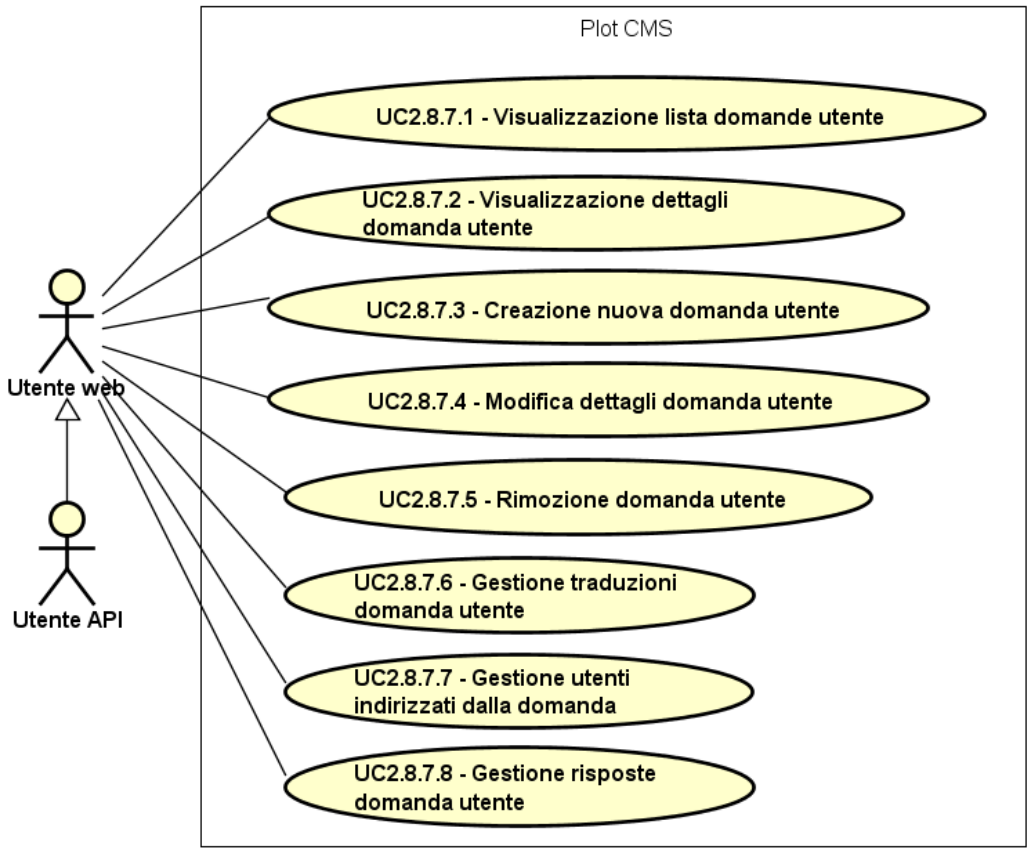


figura 3.8: Caso d'uso UC2.8.7: gestione domande utente

- **Attori**: utente web, utente API;

- **Descrizione**: un utente autenticato deve poter gestire gli aspetti di base legati alle domande utente: creazione, visualizzazione, modifica e rimozione. Inoltre deve essere possibile gestire gli utenti indirizzati dalla domanda e le traduzioni relative ad una specifica domanda. Le domande utente sono domande rivolte ai singoli utenti del gioco;
- **Precondizione**: l'utente accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema. Le domande utente hanno significato solo come figli di un post captcha, tale post deve quindi esistere per poter accedere alle funzionalità di gestione delle domande utente;
- **Flusso principale degli eventi**:
  1. L'utente autenticato può visualizzare una lista di tutte le domande collegate ad uno specifico post captcha;
  2. L'utente autenticato può visualizzare i dettagli di una specifica domanda collegata al post captcha (UC2.8.7.2);
  3. L'utente autenticato può creare un nuova domanda collegata al post captcha (UC2.8.7.3);
  4. L'utente autenticato può modificare i dettagli di una particolare domanda collegata al post di tipo captcha (UC2.8.7.4);
  5. L'utente autenticato può rimuovere una particolare domanda collegata al post di tipo captcha (UC2.8.7.5);
  6. L'utente autenticato può gestire le traduzioni della domanda collegata al post di tipo captcha (UC2.8.7.6);
  7. L'utente autenticato può gestire gli utenti indirizzati da una particolare domanda collegata al post di tipo captcha. Gli utenti indirizzati sono coloro ai quali la domanda è rivolta (UC2.8.7.7);
  8. L'utente autenticato può gestire le risposte fornite dagli utenti del gioco ad una specifica domanda utente, collegata ad un post di tipo captcha (UC2.8.7.8).
- **Postcondizione**: il sistema ha erogato le funzionalità richieste dall'utente.

3.2.24 Caso d'uso UC2.8.7.2: visualizzazione dettagli domanda utente

- **Attori**: utente web, utente API;
- **Descrizione**: l'utente autenticato deve poter visualizzare i dettagli relativi ad una specifica domanda collegata al post di tipo captcha;
- **Precondizione**: l'utente autenticato accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema. La domanda che si vuole visualizzare deve esistere;
- **Flusso principale degli eventi**:

1. L'utente autenticato visualizza il contenuto di default della domanda;
2. L'utente autenticato visualizza le aree lavorative indirizzare dalla domanda;
3. L'utente autenticato visualizza le risposte corrette previste per la domanda.

- **Postcondizione**: il sistema mostra all'utente i dettagli della domanda richiesta.

3.2.25 Caso d'uso UC2.8.7.3: creazione nuova domanda utente

- **Attori**: utente web, utente API;
- **Descrizione**: l'utente autenticato deve poter creare una nuova domanda utente, collegandola ad un post captcha;
- **Precondizione**: l'utente accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema;
- **Flusso principale degli eventi**:
  1. L'utente autenticato inserisce il contenuto di default della domanda;
  2. L'utente autenticato inserisce le aree lavorative indirizzare dalla domanda;
  3. L'utente autenticato inserisce le risposte corrette previste per la domanda.
- **Postcondizione**: il sistema crea una nuova domanda utente, collegata al post di tipo captcha, e ne mostra i dettagli all'utente.

3.2.26 Caso d'uso UC2.8.7.4: modifica dettagli domanda utente

- **Attori**: utente web, utente API;
- **Descrizione**: l'utente autenticato deve poter modificare i dettagli relativi ad una domanda utente esistente;
- **Precondizione**: l'utente accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema. La domanda che si vuole modificare deve esistere;
- **Flusso principale degli eventi**:
  1. L'utente autenticato può modificare il contenuto di default della domanda;
  2. L'utente può modificare le aree lavorative indirizzare dalla domanda;
  3. L'utente autenticato può modificare le risposte corrette previste per la domanda.
- **Postcondizione**: il sistema apporta le modifiche richieste e mostra i dettagli della domanda modificata all'utente.

3.2.27 Caso d'uso UC2.8.7.5: rimozione domanda utente

- **Attori**: utente web, utente API;
- **Descrizione**: l'utente autenticato deve poter rimuovere una domanda utente;
- **Precondizione**: l'utente autenticato accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema. La domanda che si vuole rimuovere deve esistere;
- **Flusso principale degli eventi**:
  1. L'utente autenticato rimuove una particolare domanda.
- **Postcondizione**: la domanda viene rimossa dal sistema e non può più essere recuperata. Il sistema mostra all'utente una lista delle domande utente collegate al post captcha e ancora presenti nel sistema.

3.2.28 Caso d'uso UC2.8.7.6: gestione traduzioni domanda utente

- **Attori**: utente web, utente API;
- **Descrizione**: un utente autenticato deve poter gestire gli aspetti di base legati alle traduzioni del contenuto di una domanda utente: creazione, visualizzazione, modifica e rimozione;
- **Precondizione**: l'utente accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema;
- **Flusso principale degli eventi**:
  1. L'utente autenticato può visualizzare la lista delle traduzioni previste per il contenuto della domanda utente;
  2. L'utente autenticato può visualizzare i dettagli di una particolare traduzione prevista per il contenuto della domanda utente;
  3. L'utente autenticato può creare una nuova traduzione per il contenuto della domanda utente;
  4. L'utente autenticato può modificare i dettagli di una particolare traduzione per il contenuto della domanda utente;
  5. L'utente autenticato può rimuovere una traduzione relativa al contenuto della domanda utente.
- **Postcondizione**: il sistema ha erogato le funzionalità richieste dall'utente.

3.2.29 Caso d'uso UC2.8.7.7: gestione utenti indirizzati dalla domanda

- **Attori**: utente web, utente API;
- **Descrizione**: un utente autenticato deve poter aggiungere, rimuovere e visualizzare gli utenti del gioco ai quali una particolare domanda è rivolta;

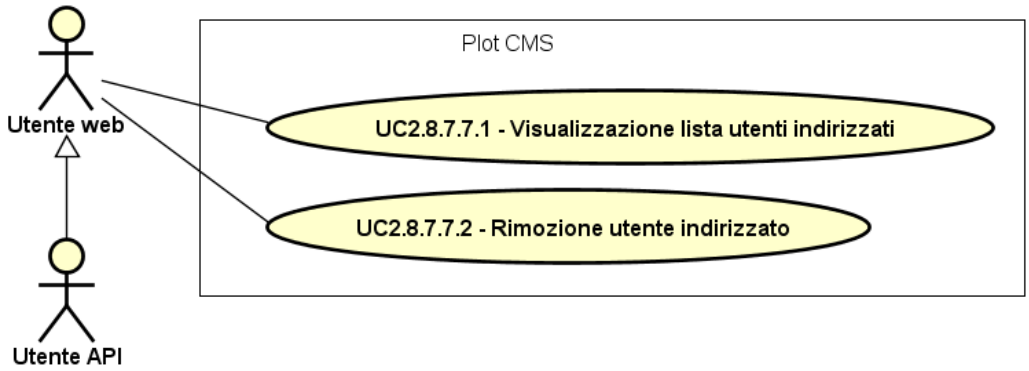


figura 3.9: Caso d'uso UC2.8.7.7: gestione utenti indirizzati dalla domanda

- **Precondizione**: l'utente accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema. Gli utenti indirizzati hanno significato solo come figli di una domanda utente, tale domanda utente deve quindi esistere ;
- **Flusso principale degli eventi**:
  1. L'utente autenticato può visualizzare una lista di tutti gli utenti indirizzati da una specifica domanda, collegata al post captcha;
  2. L'utente autenticato può rimuovere un particolare utente indirizzato dalla domanda (UC2.8.7.7.2).
- **Postcondizione**: il sistema ha erogato le funzionalità richieste dall'utente.

3.2.30 Caso d'uso UC2.8.7.7.2: rimozione utente indirizzato

- **Attori**: utente web, utente API;
- **Descrizione**: l'utente autenticato deve poter rimuovere un utente indirizzato da una specifica domanda. Rimuovere un utente indirizzato non significa rimuovere l'utente di gioco ma rimuovere l'associazione tra l'utente di gioco e la domanda, in modo che la domanda non sia più visibile all'utente di gioco;
- **Precondizione**: l'utente autenticato accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema. La utente indirizzato che si vuole rimuovere deve esistere;
- **Flusso principale degli eventi**:
  1. L'utente autenticato rimuove un utente indirizzato.
- **Postcondizione**: l'utente indirizzato viene rimosso dal sistema e non può più essere recuperato. Il sistema mostra all'utente una lista degli utenti di gioco ancora indirizzati dalla domanda.

3.2.31 Caso d'uso UC2.8.7.8: gestione risposte domanda utente

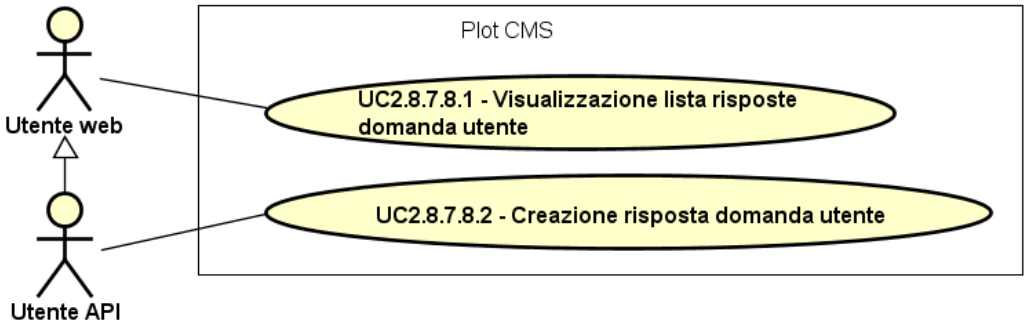


figura 3.10: Caso d'uso UC2.8.7.8: gestione risposte domanda utente

- **Attori**: utente web, utente API;
- **Descrizione**: un utente autenticato deve poter creare e visualizzare le risposte fornite dagli utenti del gioco alle domande ad essi indirizzate;
- **Precondizione**: l'utente accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema. Le risposte hanno significato solo come figli di una domanda, tale domanda deve quindi esistere;
- **Flusso principale degli eventi**:
  1. l'utente autenticato può visualizzare una lista di tutte le risposte fornite da un utente indirizzato per una specifica domanda utente (UC2.8.7.8.1);
  2. l'utente autenticato può creare una nuova risposta, fornita da uno specifico utente indirizzato ad una specifica domanda utente (UC2.8.7.8.2).
- **Postcondizione**: il sistema ha erogato le funzionalità richieste dall'utente.

3.2.32 Caso d'uso UC2.8.7.8.1: visualizzazione lista risposte domanda utente

- **Attori**: utente web, utente API;
- **Descrizione**: l'utente autenticato deve poter visualizzare una lista di tutte le risposte fornite da uno specifico utente indirizzato per una domanda utente. Ogni risposta deve riportare il contenuto della stessa ed un attributo che indichi se è corretta o meno;
- **Precondizione**: l'utente autenticato accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema;
- **Flusso principale degli eventi**:
  1. l'utente autenticato visualizza la lista delle risposte fornite.

- **Postcondizione**: il sistema mostra la lista degli utenti di gioco indirizzati dalla domanda.

3.2.33 Caso d'uso UC2.8.7.8.2: creazione risposta domanda utente

- **Attori**: utente API;
- **Descrizione**: l'utente autenticato deve poter creare una nuova risposta fornita da un utente indirizzato per una domanda utente;
- **Precondizione**: l'utente visualizza la lista degli utenti indirizzati da una specifica domanda;
- **Flusso principale degli eventi**:
  1. l'utente autenticato crea una nuova risposta.
- **Postcondizione**: il sistema mostra la lista degli utenti di gioco indirizzati dalla domanda .

3.2.34 Caso d'uso UC2.9: gestione post team

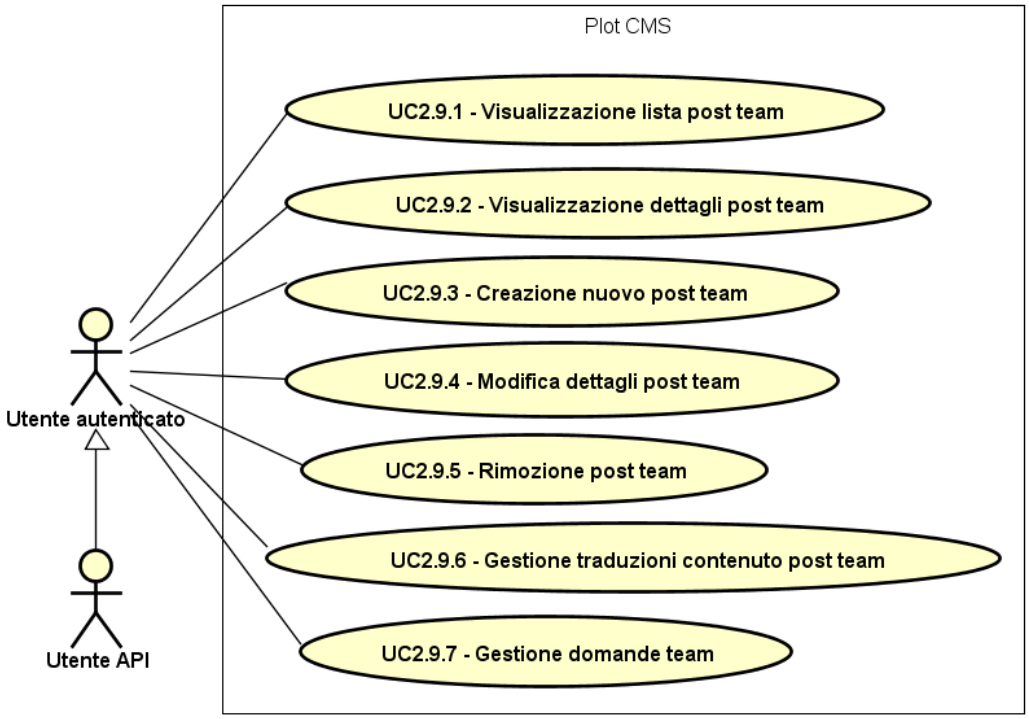


figura 3.11: Caso d'uso UC2.9: gestione post team

- **Attori**: utente web, utente API;
- **Descrizione**: un utente autenticato deve poter gestire gli aspetti di base legati ai post team: creazione, visualizzazione, modifica e rimozione. Inoltre

deve essere possibile gestire le domande legate ad uno specifico post team e le traduzioni relative al contenuto di uno specifico post team;

- **Precondizione**: l'utente accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema;
- **Flusso principale degli eventi**:
  1. L'utente autenticato può visualizzare una lista di tutti i post team relativi alla mission;
  2. L'utente autenticato può visualizzare i dettagli di uno specifico post team collegato alla mission (UC2.9.2);
  3. L'utente autenticato può creare un nuovo post team collegato alla mission (UC2.9.3);
  4. L'utente autenticato può modificare i dettagli di un particolare post team collegato alla mission (UC2.9.4);
  5. L'utente autenticato può rimuovere un particolare post team collegato alla mission (UC2.9.5);
  6. L'utente autenticato può gestire le traduzioni del contenuto di un particolare post team collegato alla mission (UC2.9.6);
  7. L'utente autenticato può gestire le domande collegate al post team (UC2.9.7).
- **Postcondizione**: il sistema ha erogato le funzionalità richieste dall'utente.

3.2.35 Caso d'uso UC2.9.2: visualizzazione dettagli post team

- **Attori**: utente web, utente API;
- **Descrizione**: l'utente autenticato deve poter visualizzare i dettagli relativi ad uno specifico post team collegato alla mission;
- **Precondizione**: l'utente autenticato accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema. Il post team che si vuole visualizzare deve esistere;
- **Flusso principale degli eventi**:
  1. L'utente autenticato visualizza il contenuto di default del post team;
  2. L'utente autenticato visualizza il grado di visibilità del post team. Questo indica se il post è visibile o meno agli utenti di gioco;
  3. L'utente autenticato visualizza il tempo limite utile per fornire la risposta corretta alle domande collegate al post .
- **Postcondizione**: il sistema mostra all'utente i dettagli del post team richiesto.

3.2.36 Caso d'uso UC2.9.3: creazione nuovo post team

- **Attori**: utente web, utente API;
- **Descrizione**: l'utente autenticato deve poter creare un nuovo post team, collegandolo alla mission;
- **Precondizione**: l'utente accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema;
- **Flusso principale degli eventi**:
  1. L'utente autenticato inserisce il contenuto di default del post team;
  2. L'utente autenticato può inserire il grado di visibilità del post captcha. Questo indica se il post è visibile o meno agli utenti di gioco;
  3. L'utente autenticato può inserire il tempo limite utile per fornire la risposta corretta alle domande collegate al post.
- **Postcondizione**: il sistema crea una nuovo post team, collegato alla mission, e ne mostra i dettagli all'utente.

3.2.37 Caso d'uso UC2.9.4: modifica dettagli post team

- **Attori**: utente web, utente API;
- **Descrizione**: l'utente autenticato deve poter modificare i dettagli relativi ad un post team esistente;
- **Precondizione**: l'utente accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema. Il post team che si vuole modificare deve esistere;
- **Flusso principale degli eventi**:
  1. L'utente autenticato può modificare il contenuto di default del post team;
  2. L'utente autenticato può modificare il grado di visibilità del post team. Questo indica se il post è visibile o meno agli utenti di gioco;
  3. L'utente autenticato può modificare il tempo limite utile per fornire la risposta corretta alle domande collegate al post.
- **Postcondizione**: il sistema apporta le modifiche richieste e mostra i dettagli del post team modificato all'utente.

3.2.38 Caso d'uso UC2.9.5: rimozione post team

- **Attori**: utente web, utente API;
- **Descrizione**: l'utente autenticato deve poter rimuovere un post team;
- **Precondizione**: l'utente autenticato accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema. Il post team che si vuole rimuovere deve esistere;

- **Flusso principale degli eventi**:
  1. L'utente autenticato rimuove un particolare post team.
- **Postcondizione**: Il post team viene rimosso dal sistema e non può più essere recuperato. Il sistema mostra all'utente una lista dei post team collegati alla mission e ancora presenti nel sistema.

3.2.39 Caso d'uso UC2.9.6: gestione traduzioni contenuto post team

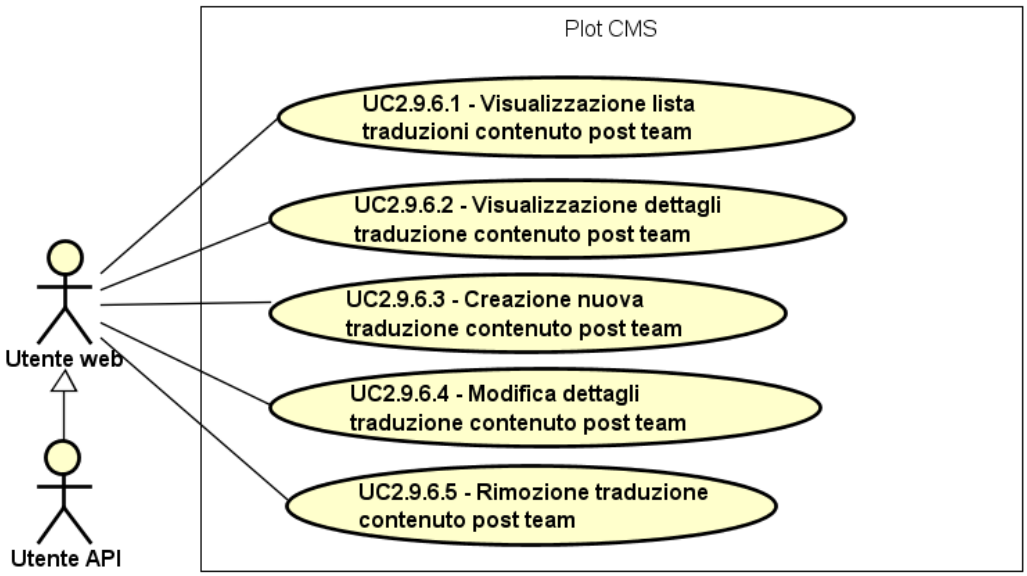


figura 3.12: Caso d'uso UC2.9.6: gestione traduzioni contenuto post team

- **Attori**: utente web, utente API;
- **Descrizione**: un utente autenticato deve poter gestire gli aspetti di base legati alle traduzioni del contenuto di un post team: creazione, visualizzazione, modifica e rimozione;
- **Precondizione**: l'utente accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema;
- **Flusso principale degli eventi**:
  1. L'utente autenticato può visualizzare la lista delle traduzioni previste per il contenuto del post team;
  2. L'utente autenticato può visualizzare i dettagli di una particolare traduzione prevista per il contenuto del post team;
  3. L'utente autenticato può creare una nuova traduzione per il contenuto del post team ;

4. L'utente autenticato può modificare i dettagli di una particolare traduzione per il contenuto del post team;
5. L'utente autenticato può rimuovere una traduzione relativa al contenuto del post team.

- **Postcondizione**: il sistema ha erogato le funzionalità richieste dall'utente.

3.2.40 Caso d'uso UC2.9.7: gestione domande team

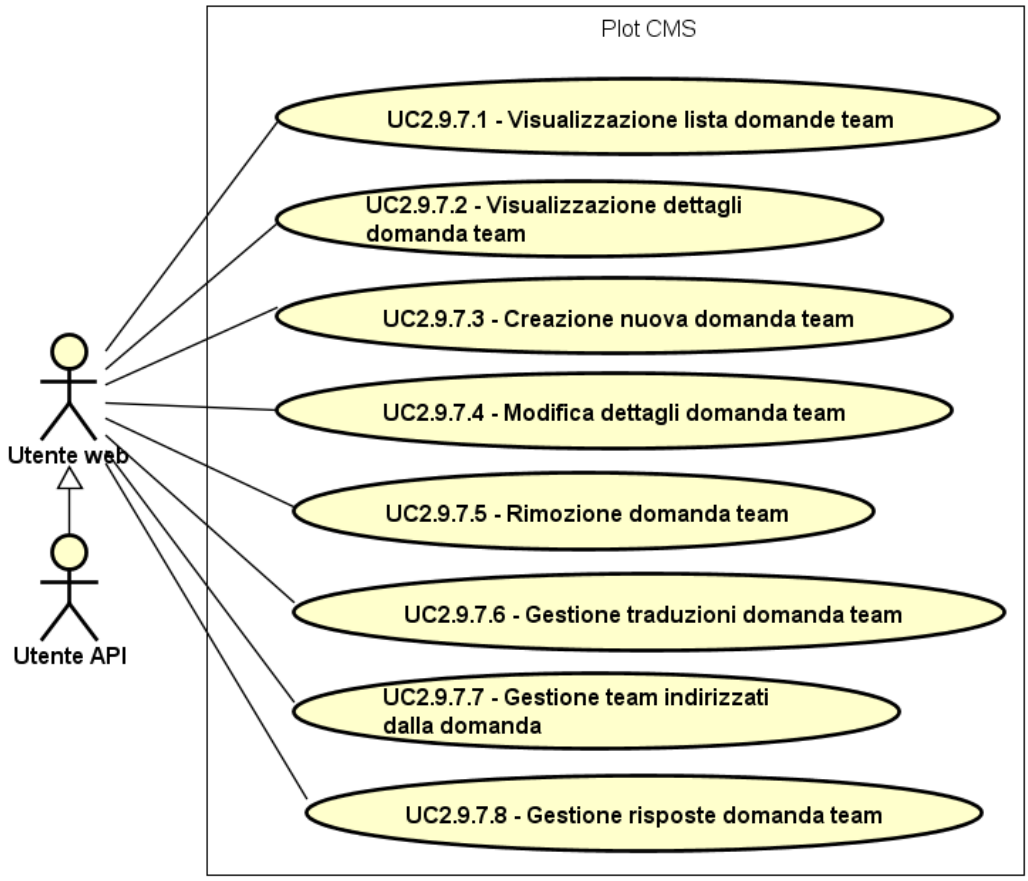


figura 3.13: Caso d'uso UC2.9.7: gestione domande team

- **Attori**: utente web, utente API;
- **Descrizione**: un utente autenticato deve poter gestire gli aspetti di base legati alle domande rivolte ai team: creazione, visualizzazione, modifica e rimozione. Inoltre deve essere possibile gestire i team indirizzati dalla domanda e le traduzioni relative ad una specifica domanda;
- **Precondizione**: l'utente accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema. Le domande team hanno significato solo come figli di un post team, tale post deve quindi esistere per poter accedere alle funzionalità di gestione delle domande team;

- **Flusso principale degli eventi**:
  1. L'utente autenticato può visualizzare una lista di tutte le domande collegate ad uno specifico post team;
  2. L'utente autenticato può visualizzare i dettagli di una specifica domanda collegata al post team (UC2.9.7.2);
  3. L'utente autenticato può creare un nuova domanda collegata al post team (UC2.9.7.3);
  4. L'utente autenticato può modificare i dettagli di una particolare domanda collegata al post di tipo team (UC2.9.7.4);
  5. L'utente autenticato può rimuovere una particolare domanda collegata al post di tipo team (UC2.9.7.5);
  6. L'utente autenticato può gestire le traduzioni della domanda collegata al post di tipo team (UC2.9.7.6);
  7. L'utente autenticato può gestire gli utenti indirizzati da una particolare domanda collegata al post di tipo team. Gli utenti indirizzati sono coloro ai quali la domanda è rivolta (UC2.9.7.7);
  8. L'utente autenticato può gestire le risposte fornite dagli utenti del gioco ad una specifica domanda team, collegata ad un post di tipo team (UC2.9.7.8).
- **Postcondizione**: il sistema ha erogato le funzionalità richieste dall'utente.

3.2.41 Caso d'uso UC2.9.7.2: visualizzazione dettagli domanda team

- **Attori**: utente web, utente API;
- **Descrizione**: l'utente autenticato deve poter visualizzare i dettagli relativi ad una specifica domanda collegata al post di tipo team;
- **Precondizione**: l'utente autenticato accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema. La domanda che si vuole visualizzare deve esistere;
- **Flusso principale degli eventi**:
  1. L'utente autenticato visualizza il contenuto di default della domanda;
  2. L'utente autenticato visualizza le risposte corrette previste per la domanda.
- **Postcondizione**: il sistema mostra all'utente i dettagli della domanda richiesta.

3.2.42 Caso d'uso UC2.9.7.3: creazione nuova domanda team

- **Attori**: utente web, utente API;
- **Descrizione**: l'utente autenticato deve poter creare una nuova domanda team, collegandola ad un post di tipo team;

- **Precondizione**: l'utente accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema;
- **Flusso principale degli eventi**:
  1. L'utente autenticato inserisce il contenuto di default della domanda;
  2. L'utente autenticato inserisce le risposte corrette previste per la domanda.
- **Postcondizione**: il sistema crea una nuova domanda collegata al post di tipo team, e ne mostra i dettagli all'utente.

3.2.43 Caso d'uso UC2.9.7.4: modifica dettagli domanda team

- **Attori**: utente web, utente API;
- **Descrizione**: l'utente autenticato deve poter modificare i dettagli relativi ad una domanda di tipo team esistente;
- **Precondizione**: l'utente accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema. La domanda che si vuole modificare deve esistere;
- **Flusso principale degli eventi**:
  1. L'utente autenticato può modificare il contenuto di default della domanda;
  2. L'utente autenticato può modificare le risposte corrette previste per la domanda.
- **Postcondizione**: il sistema apporta le modifiche richieste e mostra i dettagli della domanda modificata all'utente.

3.2.44 Caso d'uso UC2.9.7.5: rimozione domanda team

- **Attori**: utente web, utente API;
- **Descrizione**: l'utente autenticato deve poter rimuovere una domanda di tipo team;
- **Precondizione**: 'utente autenticato accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema. La domanda che si vuole rimuovere deve esistere;
- **Flusso principale degli eventi**:
  1. L'utente autenticato rimuove una particolare domanda.
- **Postcondizione**: la domanda viene rimossa dal sistema e non può più essere recuperata. Il sistema mostra all'utente una lista delle domande collegate al post di tipo team ancora presenti nel sistema.

3.2.45 Caso d'uso UC2.9.7.6: gestione traduzioni domanda team

- **Attori**: utente web, utente API;
- **Descrizione**: un utente autenticato deve poter gestire gli aspetti di base legati alle traduzioni del contenuto di una domanda di tipo team: creazione, visualizzazione, modifica e rimozione;
- **Precondizione**: l'utente accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema;
- **Flusso principale degli eventi**:
  1. L'utente autenticato può visualizzare la lista delle traduzioni previste per il contenuto della domanda di tipo team;
  2. L'utente autenticato può visualizzare i dettagli di una particolare traduzione prevista per il contenuto della domanda di tipo team;
  3. 'utente autenticato può creare una nuova traduzione per il contenuto della domanda di tipo team;
  4. L'utente autenticato può modificare i dettagli di una particolare traduzione per il contenuto della domanda di tipo team;
  5. L'utente autenticato può rimuovere una traduzione relativa al contenuto della domanda di tipo team.
- **Postcondizione**: il sistema ha erogato le funzionalità richieste dall'utente.

3.2.46 Caso d'uso UC2.9.7.7: gestione team indirizzati dalla domanda

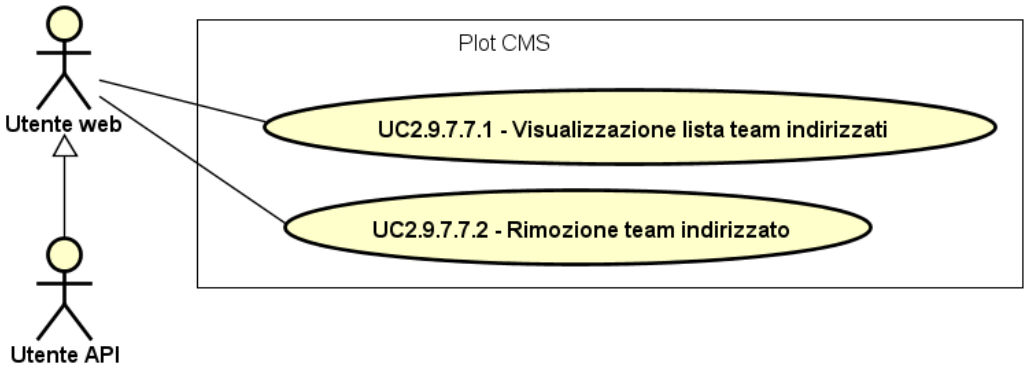


figura 3.14: Caso d'uso UC2.9.7.7: gestione team indirizzati dalla domanda

- **Attori**: utente web, utente API;
- **Descrizione**: un utente autenticato deve poter aggiungere, rimuovere e visualizzare i team ai quali una particolare domanda è rivolta;

- **Precondizione**: l'utente accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema. I team indirizzati hanno significato solo come figli di una domanda di tipo team, tale domanda deve quindi esistere;
- **Flusso principale degli eventi**:
  1. L'utente autenticato può visualizzare una lista di tutti i team indirizzati da una specifica domanda, collegata al post di tipo team (UC2.9.7.7.1);
  2. L'utente autenticato può rimuovere un particolare team indirizzato dalla domanda (UC2.9.7.7.2).
- **Postcondizione**: il sistema ha erogato le funzionalità richieste dall'utente.

3.2.47 Caso d'uso UC2.9.7.7.1: visualizzazione lista team indirizzati

- **Attori**: utente web, utente API;
- **Descrizione**: l'utente autenticato deve poter visualizzare una lista di tutti i team ai quali una specifica domanda (collegata ad un post di tipo team) è rivolta;
- **Precondizione**: l'utente autenticato accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema;
- **Flusso principale degli eventi**:
  1. l'utente visualizza la lista dei team indirizzati da una specifica domanda.
- **Postcondizione**: il sistema mostra la lista dei team indirizzati dalla domanda.

3.2.48 Caso d'uso UC2.9.7.7.2: rimozione team indirizzato

- **Attori**: utente web, utente API;
- **Descrizione**: l'utente autenticato deve poter rimuovere un team indirizzato da una specifica domanda. Rimuovere un team indirizzato non significa rimuovere il team dal gioco ma rimuovere l'associazione tra il team e la domanda, in modo che la domanda non sia più visibile la team;
- **Precondizione**: l'utente autenticato accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema. Il team indirizzato che si vuole rimuovere deve esistere;
- **Flusso principale degli eventi**:
  1. L'utente autenticato rimuove un team indirizzato.
- **Postcondizione**: il team indirizzato viene rimosso dal sistema e non può più essere recuperato. Il sistema mostra all'utente una lista dei team ancora indirizzati dalla domanda.

3.2.49 Caso d'uso UC2.9.7.8: gestione risposte domanda team

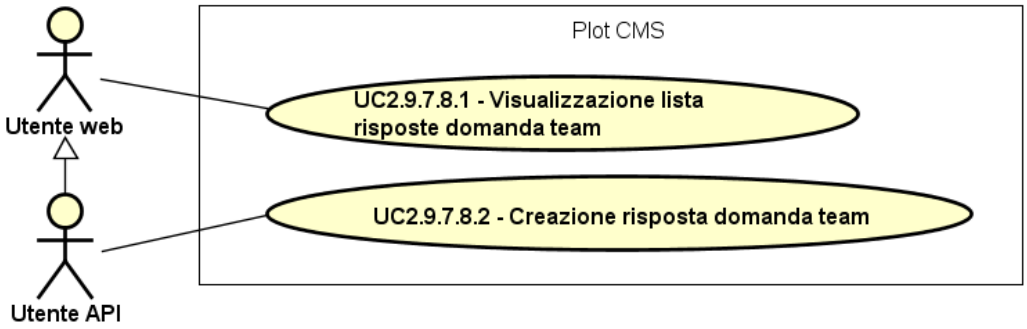


figura 3.15: Caso d'uso UC2.9.7.8: gestione risposte domanda team

- **Attori**: utente web, utente API;
- **Descrizione**: un utente autenticato deve poter creare e visualizzare le risposte fornite dai team alle domande ad essi indirizzate;
- **Precondizione**: l'utente accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema. Le risposte hanno significato solo come figli di una domanda, tale domanda deve quindi esistere;
- **Flusso principale degli eventi**:
  1. l'utente autenticato può visualizzare una lista di tutte le risposte fornite da un team indirizzato per una specifica domanda di tipo team (UC2.9.7.8.1);
  2. l'utente autenticato può creare una nuova risposta, fornita da uno specifico team indirizzato ad una specifica domanda udi tipo team (UC2.9.7.8.2).
- **Postcondizione**: il sistema ha erogato le funzionalità richieste dall'utente.

3.2.50 Caso d'uso UC2.9.7.8.1: visualizzazione lista risposte domanda team

- **Attori**: utente web, utente API;
- **Descrizione**: l'utente autenticato deve poter visualizzare una lista di tutte le risposte fornite da uno specifico team indirizzato per una domanda di tipo team. Ogni risposta deve riportare il contenuto della stessa ed un attributo che indichi se è corretta o meno;
- **Precondizione**: l'utente autenticato accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema;
- **Flusso principale degli eventi**:
  1. l'utente autenticato visualizza la lista delle risposte fornite.

- **Postcondizione**: il sistema mostra la lista degli utenti di gioco indirizzati dalla domanda.

3.2.51 Caso d'uso UC2.9.7.8.2: creazione risposta domanda team

- **Attori**: utente API;
- **Descrizione**: l'utente autenticato deve poter creare una nuova risposta fornita da un team indirizzato per una domanda udi tipo team;
- **Precondizione**: l'utente visualizza la lista degli utenti indirizzati da una specifica domanda;
- **Flusso principale degli eventi**:
  1. l'utente autenticato crea una nuova risposta.
- **Postcondizione**: il sistema mostra la lista dei team in gioco indirizzati dalla domanda.

3.2.52 Caso d'uso UC7: visualizzazione errore login

- **Attori**: utente non autenticato;
- **Descrizione**: viene visualizzato un messaggio d'errore ad indicare che l'autenticazione dell'utente non è andata a buon fine. Questo può accadere nel caso in cui l'utente inserisca delle credenziali non riconosciute dal sistema;
- **Precondizione**: l'utente non è già autenticato, accede alla piattaforma e utilizza delle credenziali non riconosciute per autenticarsi al sistema;
- **Flusso principale degli eventi**:
  1. L'utente visualizza un messaggio d'errore relativo alla fallita autenticazione.
- **Postcondizione**: il messaggio d'errore viene mostrato all'utente.

Capitolo 4

Progettazione e codifica

4.1 Progettazione del database

Dall'analisi dei requisiti del database è stato possibile ricavare lo schema del database da implementare. Nel presente documento non è possibile riportare completamente tale schema a causa delle sue grandi dimensioni, ne verranno dunque descritti e riportati i dettagli salienti.

Per ottenere lo schema finale è stato innanzitutto modellato uno schema concettuale di base, tale schema è stato successivamente ristrutturato in modo da:

- Eliminare le ridondanze$_G$ non necessarie;
- Eliminare le gerarchie: MySQL$_G$ non consente la rappresentazione diretta di una gerarchia, si deve quindi trasformare tale costrutto in modo che possa essere implementato;
- Partizionare le relazioni N-N: tali relazioni sono da evitare in quanto portano ad avere valori duplicati in entrambe le entità che vi partecipano. La presenza di valori duplicati implica una maggiore complessità nella risoluzione delle query e, quindi, un abbassamento generale delle performance. Per ovviare a questo problema sono state introdotte varie *entità aggregate* per dividere una relazione N-N in due relazioni 1-N o 0-N.

4.1.1 Dizionario delle entità

La tabella seguente documenta il significato delle entità rappresentate nello schema finale del database, in modo da evitare quanto più possibile situazioni di ambiguità.

Ogni entità è dotata degli attributi: id, created_at, updated_at. Questi non sono stati riportati nel dizionario delle entità, in quanto sono stati messi in risalto solo gli attributi caratteristici di ogni entità.

Entità	**Descrizione**	**Attributi**
users	Utenti iscritti a solo scopo di gioco	name, surname, email, password, remember_token
external_auth	Token di autenticazione fornito da un servizio esterno	external_uid
admins	Utenti con accesso al solo CMS	name, surname, email, password, remember_token
invitations	Inviti mandati e ricevuti dagli utenti	
logs	Log generati dalle richieste HTTP effettuate dagli user nel gioco	method, url, user_ip_address, user_agent, additional_data
regions	Aree geografiche previste dal gioco	name
timezones	Fusi orari previsti dal gioco	code, name, offset
business_roles	Ambiti lavorativi previsti dal gioco	name
languages	Lingue supportate dal gioco	locale, code
translatable_contents	I contenuti traducibili del gioco, ovvero quei contenuti che possono essere tradotti	default_text
translations	Tutte le traduzioni disponibili relativamente ai contenuti traducibili presenti nel gioco	text
teams	Squadre presenti nel gioco	name, cover_image
missions	Mission previste dal gioco	cover_image, is_visible, start_date, end_date
text_posts	I post di tipo testo	is_visible
captcha_posts	I post di tipo captcha	is_visible

Entità	Descrizione	Attributi
team_posts	I post di tipo team	is_visible, time_limit
user_questions	Domande rivolte ai singoli utenti	
user_correct_answers	Le risposte corrette previste per le domande rivolte ai singoli utenti	content
user_questions_business_roles	Associazione tra le domande rivolte ai singoli utenti e le aree lavorative alle quali sono indirizzate. In questo modo, ogni possibile user_question può essere indirizzata a tutti quegli utenti aventi il medesimo business_role	
made_user_questions	Associazione tra gli utenti e le domande ad essi effettivamente rivolte	
user_answers	Risposte fornite dagli utenti in merito alle domande ad essi effettivamente rivolte	content, is_correct
team_questions	Domande rivolte ai team	
team_correct_answers	Le risposte corrette previste per le domande rivolte ai team	content
made_team_questions	Associazione tra i team e le domande ad essi effettivamente rivolte	
team_answers	Risposte fornite dai team in merito alle domande ad essi effettivamente rivolte	content, is_correct
scores	Punteggi ottenuti dai team nel completare le varie mission	points

tabella 4.1: Dizionario delle entità

4.1.2 Dizionario delle relazioni

La tabella seguente documenta il significato delle relazioni rappresentate nello schema finale del database, in modo da evitare quanto più possibile situazioni di ambiguità. La colonna "Entità" riporta le entità partecipanti alla relazione, accompagnate dalla loro cardinalità.

Relazione	**Descrizione**	**Entità**
have	Associa un utente di gioco con il proprio token di autenticazione esterno	users (0,1) external_auth (1,1)
do	Associa un utente di gioco con gli inviti da esso spediti	users (0,N) invitations (1,1)
receive	Associa un utente di gioco con gli inviti da esso ricevuti	users (0,N) invitations (1,1)
produce	Associa un utente di gioco con i log prodotti dalle sue richieste HTTP	users (0,N) logs (1,1)
belong	Associa un utente di gioco con l'area geografica alla quale appartiene	users (0,1) regions (0,N)
belong	Associa un'area geografica con il fuso orario al quale appartiene	regions (1,1) timezones (0,N)
have	Associa un utente di gioco con l'ambito lavorativo al quale appartiene	users (0,1) business_roles (0,N)
speak	Associa un utente di gioco alla sua lingua preferita	users (1,1) languages (0,N)
address	Associa ogni traduzione alla propria lingua	translations (1,1) languages (0,N)
have	Associa un contenuto traducibile con tutte le sue traduzioni disponibili	translatable_contents (0,N) translations (1,1)
have	Associa un utente al team al quale appartiene	users (0,1) teams (0,N)
name	Associa una mission al contenuto traducibile che rappresenta il nome della mission stessa	missions (1,1) translatable_contents (1,1)
have	Associa un post di tipo testo con la mission alla quale appartiene	text_posts (1,1) missions (0,N)
have	Associa un post di tipo captcha con la mission alla quale appartiene	captcha_posts (1,1) missions (0,N)
have	Associa un post di tipo team con la mission alla quale appartiene	team_posts (1,1) missions (0,N)
content	Associa un post di tipo testo al contenuto traducibile che rappresenta il corpo del post stesso	text_posts (1,1) translatable_contents (1,1)
abstract	Associa un post di tipo captcha al contenuto traducibile che rappresenta il corpo del post stesso	captcha_posts (1,1) translatable_contents (1,1)

Relazione	Descrizione	Entità
abstract	Associa un post di tipo team al contenuto traducibile che rappresenta il corpo del post stesso	team_posts (1,1) translatable_contents (1,1)
choose	Associa un post di tipo captcha al bacino delle domande che possono essere rivolte ai singoli utenti	captcha_posts (0,N) user_questions (1,1)
choose	Associa un post di tipo team al bacino delle domande che possono essere rivolte ai team	team_posts (0,N) team_questions (1,1)
content	Associa una domanda utente al contenuto traducibile che rappresenta il corpo della domanda stessa	user_questions (1,1) translatable_contents (1,1)
content	Associa una domanda di tipo team al contenuto traducibile che rappresenta il corpo della domanda stessa	team_questions (1,1) translatable_contents (1,1)
have	Associa una domanda utente alle risposte corrette previste per la domanda stessa	user_questions (1,N) user_correct_answers (1,1)
aggregate	Collega una associazione tra domanda utente e ambito lavorativo con la rispettiva domanda utente. Questa relazione nasce dalla scomposizione della relazione N-N che legava una domanda utente con gli ambiti lavorativi ai quali è rivolta	user_questions_business_roles (1,1) user_questions (0,N)
aggregate	Collega una associazione tra domanda utente e ambito lavorativo con il rispettivo ambito lavorativo. Questa relazione nasce dalla scomposizione della relazione N-N che legava una domanda utente con gli ambiti lavorativi ai quali è rivolta	user_questions_business_roles (1,1) business_roles (0,N)
aggregate	Collega una associazione tra domanda utente e l'utente al quale è stata rivolta con la rispettiva domanda utente. Questa relazione nasce dalla scomposizione della relazione N-N che legava una domanda utente con l'utente al quale è rivolta	made_user_questions (1,1) user_questions (0,N)
aggregate	Collega una associazione tra domanda utente e l'utente al quale è stata rivolta con l'utente stesso. Questa relazione nasce dalla scomposizione della relazione N-N che legava una domanda utente con l'utente al quale è rivolta	made_user_questions (1,1) users (0,N)

Relazione	Descrizione	Entità
aggregate	Collega una associazione tra domanda di tipo team e il team al quale è stata rivolta con la rispettiva domanda di tipo team. Questa relazione nasce dalla scomposizione della relazione N-N che legava una domanda di tipo team con il team al quale è rivolta	made_team_questions (1,1) team_questions (0,N)
aggregate	Collega una associazione tra domanda utente e l'utente al quale è stata rivolta con il team stesso. Questa relazione nasce dalla scomposizione della relazione N-N che legava una domanda di tipo team con il team al quale è rivolta	made_team_questions (1,1) teams (0,N)
refer	Collega una associazione tra domanda utente e l'utente al quale è stata rivolta con le risposte fornite per tale domanda dall'utente coinvolto	made_user_questions (0,N) user_answers (1,1)
refer	Collega una associazione tra domanda di tipo team e il team al quale è stata rivolta con le risposte fornite per tale domanda dal team coinvolto	made_team_questions (0,N) team_answers (1,1)
give	Associa una risposta fornita per una domanda utente all'utente che l'ha effettivamente fornita	user_answers (1,1) users (0,N)
give	Associa una risposta fornita per una domanda di tipo team al team che l'ha effettivamente fornita	team_answers (1,1) teams (0,N)
unveil	Collega una associazione tra domanda di tipo team e il team al quale è stata rivolta con l'utente che ha svelato la domanda avviando il timer	made_team_questions (0,1) users (0,N)
have	Associa un team ai punteggi ottenuti nella risoluzione delle varie mission	teams (0,N) scores (1,1)
have	Associa un punteggio con la mission nella quale è stato ottenuto	scores (1,1) missions (0,N)

tabella 4.2: Dizionario delle relazioni

4.1.3 Regole di vincolo

Alcuni vincoli e proprietà non sono immediatamente deducibili dallo schema del database, è stato quindi deciso di mettere per iscritto tali proprietà in modo da aumentare la chiarezza della progettazione:

1. Ogni utente può inviare al più un invito a qualsiasi altro utente;

2. Un utente può scegliere un'unica lingua preferita;
3. Un utente può afferire al più ad un ambito lavorativo;
4. Un utente può appartenere al più ad un'area geografica;
5. Uno punteggio deve appartenere ad un unico team e può riferirsi ad un'unica mission completata da tale team;
6. Un post deve appartenere ad un'unica mission;
7. Una domanda utente deve appartenere ad un unico post di tipo captcha;
8. Una domanda di tipo team deve appartenere ad un unico post di tipo team;
9. Una domanda di tipo team può essere svelata al più da un utente;
10. Un'area geografica deve appartenere ad un unico fuso orario;
11. Una risposta corretta deve riferirsi ad un'unica domanda;
12. Una risposta deve riferirsi ad un'unica domanda;
13. Un log deve riferirirsi ad un unico utente;
14. Una traduzione deve riferirsi ad un unico contenuto traducibile e ad un'unica lingua.

4.2 Progettazione ed implementazione del CMS

4.2.1 Descrizione dell'architettura

Il framework Laravel agevola l'adozione del pattern architetturale **model-view-controller (MVC)**, che impone la suddivisione della *business logic* dalla *presentation logic*. Nelle applicazioni Laravel, la business logic coincide spesso con modelli di dati persistenti in un database mentre la presentation logic si concretizza in una serie di pagine web.
Il pattern MVC prevede tre componenti principali: **model**, **view** e **controller**.

Model

Il model rappresenta il dominio dell'applicazione e contiene l'astrazione di tutti i dati necessari al buon funzionamento della piattaforma. Nello specifico, esiste un'istanza di model per ogni entità prevista dal database collegato alla piattaforma, questo permette ed agevola l'interazione con il database attraverso l'ORM$_G$ Eloquent.
È compito del model notificare alla view eventuali cambiamenti avvenuti a livello del database, in modo da mostrare all'utente sempre e solo i dati aggiornati.
Le convenzioni del framework Laravel consigliano di posizionare le istanze del model nella directory `app/Models`.

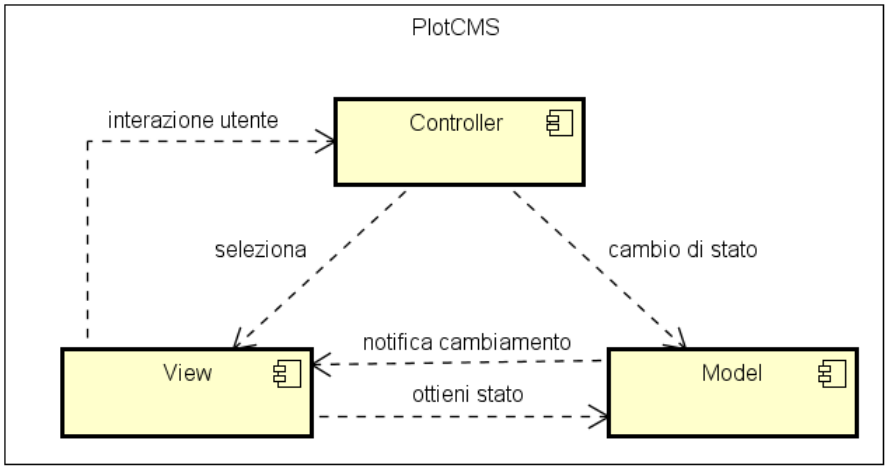


figura 4.1: Diagramma delle componenti del pattern MVC

View

La view è la rappresentazione visuale del model, cattura gli input dell'utente e ne delega l'elaborazione al controller. Nello specifico, l'interfaccia utente è composta da diverse pagine web che possono essere utilizzate dagli amministratori del CMS_G per accedere alle varie funzionalità da esso offerte.
Le convenzioni del framework Laravel consigliano di posizionare le istanze della view nella directory `resources/view`.

Controller

Il controller coordina e crea un collegamento tra view e model, è responsabile quindi dell'elaborazione dell'input e dell'interazione con il model.
Le convenzioni del framework Laravel consigliano di posizionare le istanze del controller nella directory `app/Http/Controllers`.

4.3 Descrizione delle classi

Di seguito sono descritte le classi e le interfacce più significative che compongono il CMS_G, suddivise in base alla componente alla quale appartengono.
Le classi appartenenti alla stessa componente hanno spesso implementazione simile in quanto, di base, permettono il compimento delle azioni $CRUD_G$ su risorse di tipo diverso. Per evitare di appesantire la documentazione, è stato scelto di fornire un esempio di classe laddove la descrizione di tutte le classi avrebbe reso il testo fortemente ridondante.
Le classi appartenenti al framework sono state evidenziate in arancio nei diagrammi che le prevedono, in modo da poterle individuare facilmente.

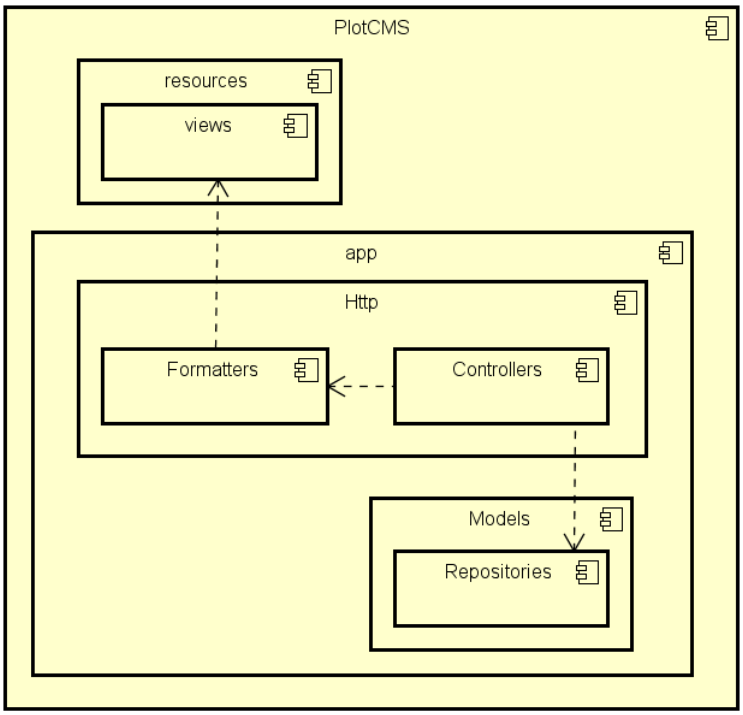


figura 4.2: Organizzazione delle principali componenti della piattaforma

4.3.1 app::Models

Le classi appartenenti a questa componente forniscono un'astrazione delle entità presenti nel database collegato alla piattaforma e sono utilizzate per interagire con esse.
L'ORM$_G$ *Eloquent* richiede che tutti i model espongano particolari metodi utili a descrivere le relazioni esistenti tra le entità rappresentate dai model stessi a livello di database.

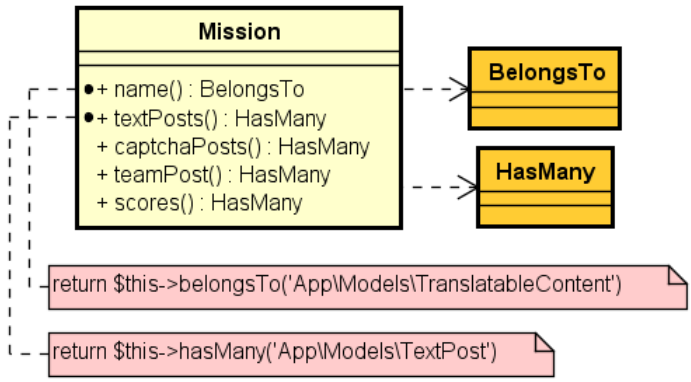


figura 4.3: Diagramma della classe Mission

Ad esempio, la classe Mission prevede i seguenti metodi:

- `name()`: rappresenta la relazione con cardinalità 1-1 tra una mission ed il contenuto traducibile che corrisponde al nome della mission stessa;
- `textPosts()`: rappresenta la relazione con cardinalità 0-N tra una mission ed i post di tipo testo collegati alla stessa;
- `captchaPosts()`: rappresenta la relazione con cardinalità 1-N tra una mission ed i post di captcha testo collegati alla stessa;
- `teamPosts()`: rappresenta la relazione con cardinalità 1-N tra una mission ed i post di tipo team collegati alla stessa;
- `scores()`: rappresenta la relazione con cardinalità 1-N tra una mission ed i punteggi totalizzati dai team relativamente alla mission stessa.

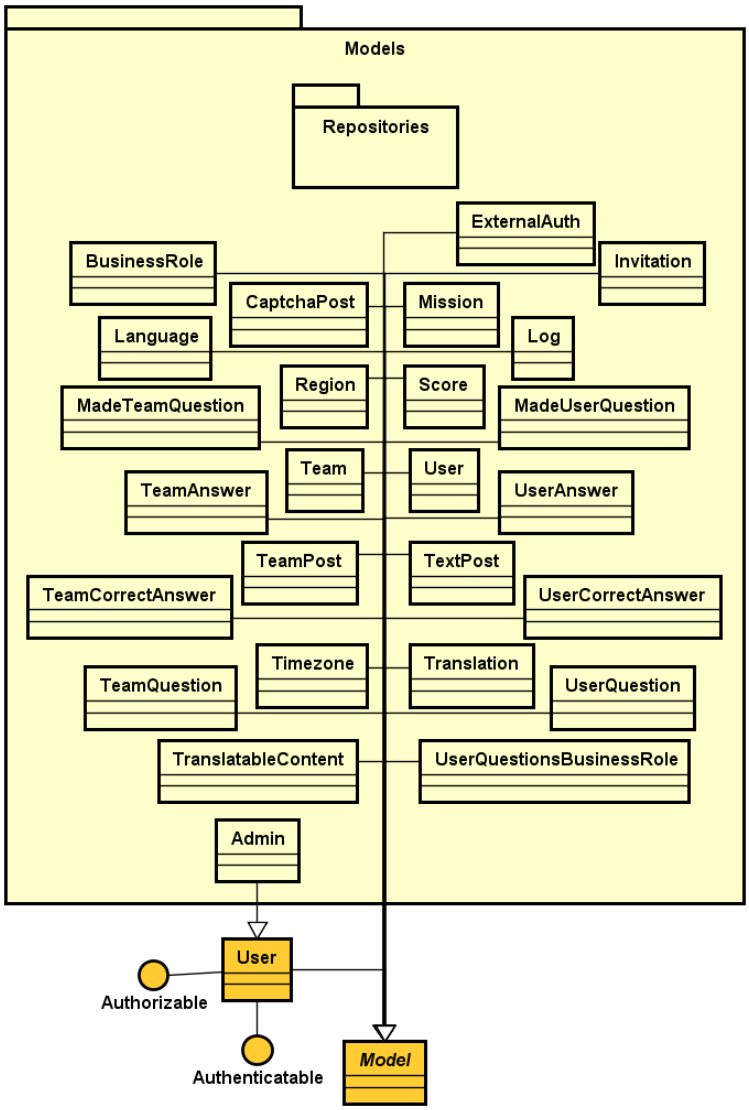


figura 4.4: Diagramma delle classi della componente Models

Maggiori dettagli sui metodi esposti dall'ORM$_G$ Eloquent per rappresentare le relazioni tra model possono essere recuperati nella documentazione online ufficiale del framework Laravel (*Laravel Documentation*. URL: https://laravel.com/docs/).

app::Models::Repositories

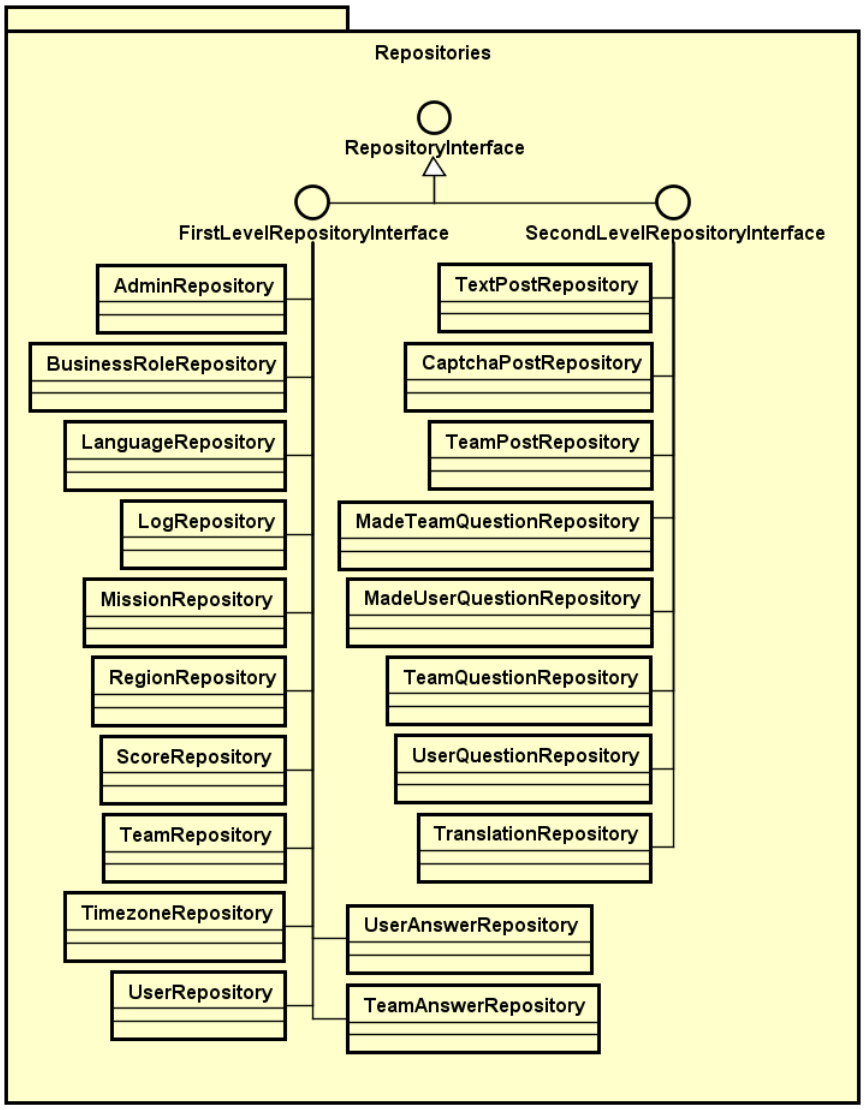


figura 4.5: Diagramma delle classi della componente Repositories

Le classi e le interfacce appartenenti a questa componente sono utilizzate per implementare il design pattern **Repository**, utile per disaccoppiare la *business logic* dal sistema di persistenza dati effettivamente utilizzato. In questo modo le istanze del controller possono effettuare query in modo dichiarativo sulle istanze del model, lasciando che siano i repository ad occuparsi dei dettagli implementativi delle query stesse.
Concettualmente, un repository incapsula l'insieme delle operazioni disponibili sulle

entità di un database, così da fornire un modo più *object-oriented* per accedervi.

Le istanze di repository possono essere fondamentalmente di due tipi, in base al model che devono trattare:

- **First level**: se l'istanza del repository è collegata ad un model la cui esistenza è autoesplicativa, ovvero non necessita di un secondo model;
- **Second level**: se l'istanza del repository è collegata ad un model che ha senso di esistere solo come figlio di un secondo model. Un esempio di questa tipologia è TextPostRepository, collegato al model TextPost le cui istanze sono significative solo come figlie di un'istanza di Mission.

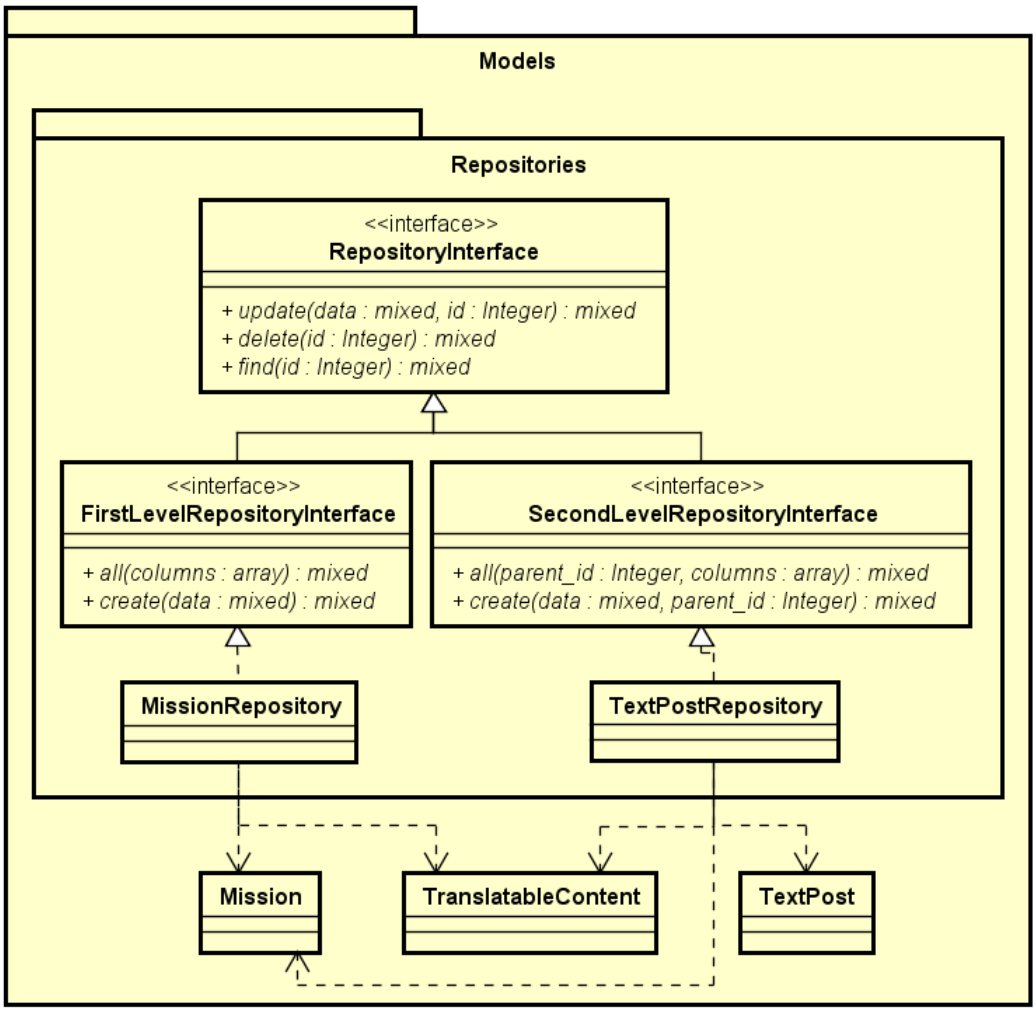


figura 4.6: Diagramma delle classi MissionRepository e TextPostRepository

Per modellare la situazione sopra descritta sono state usate delle interfacce:

- **Repositories::RepositoryInterface**: espone metodi per effettuare l'aggiornamento, la rimozione o il recupero di particolari risorse. Tali metodi sono indipendenti dal tipo di repository;
- **Repositories::FirstLevelRepositoryInterface**: espone metodi per effettuare il recupero e la creazione di risorse che non prevedono una risorsa padre. Estende `RepositoryInterface`;
- **Repositories::SecondLevelRepositoryInterface**: espone metodi per effettuare il recupero e la creazione di risorse che necessitano di una risorsa padre. Estende `RepositoryInterface`.

Un esempio di repository *first level* è MissionRepository, mentre un esempio di repository *second level* è TextPostRepository. Il secondo differisce dal primo a causa del parametro `parent_id`, necessario per individuare il padre della risorsa da creare e/o recuperare.

4.3.2 resources::views

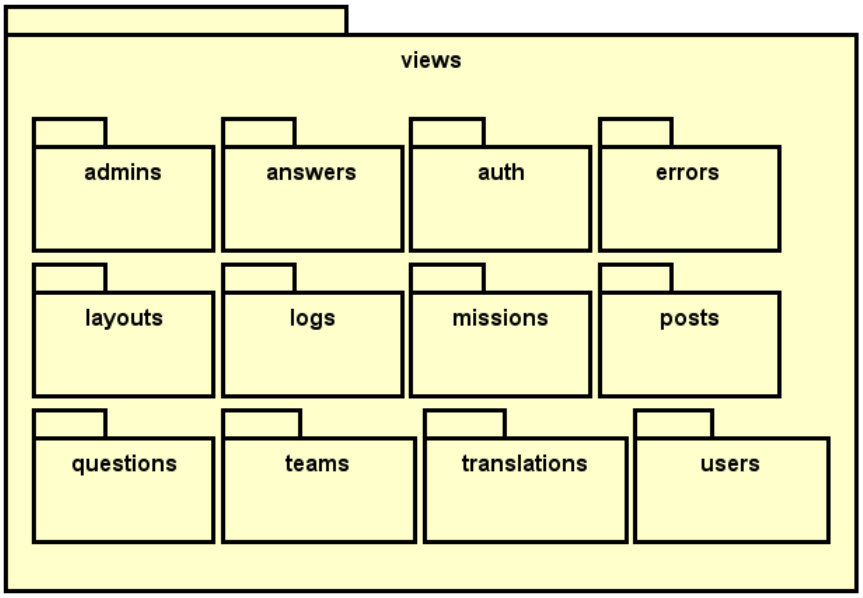


figura 4.7: Diagramma delle classi della componente views

I file appartenenti a questa componente costituiscono l'interfaccia utente del CMS_G, accessibile attraverso un browser web. L'interfaccia utente è stata ottenuta utilizzando i widget messi a disposizione dal template Bootstrap "Gentelella", liberamente disponibile online all'indirizzo `https://github.com/puikinsh/gentelella`. La scelta per la realizzazione dell'interfaccia utente è ricaduta sull'utilizzo di un template in modo da poter contenere i tempi di progettazione, senza dover per questo rinunciare alla qualità visiva e funzionale.

Per aumentare la modularizzazione delle view e la loro manutenibilità è stato utilizzato **Blade**: il *template system* offerto dal framework Laravel. Grazie a Blade è possibile:

- Definire dei *template* che fungano da base per la costruzione di diverse view;
- Utilizzare delle scorciatoie che agevolano azioni comunemente effettuate dalle view sui dati che devono mostrare, come stampare il valore di una variabile o scorrere una struttura dati.

L'utilizzo dei template ha reso possibile introdurre il concetto di ereditarietà, seppur abbastanza superficiale, nell'ambito HTML che ne è storicamente sprovvisto.
I file contenenti le view sono poi stati suddivisi sulla base del model al quale sono collegati. Ad esempio, nella sotto-componente **views::missions** sono contenuti i file:

- **all.blade**: struttura della view che mostra la lista di tutte le mission previste dal gioco;
- **create.blade**: struttura della view che presenta il form utile per creare una nuova mission;
- **edit.blade**: struttura della view che mostra il form utile a modificare i dettagli di una mission esistente;
- **show.blade**: struttura della view che mostra i dettagli di una mission esistente.

La sotto-componente più importante è **views::layouts** che contiene il file **app.blade**. Questo file definisce la base di ogni pagina web, una struttura a tre pannelli che prevede:

- Un menù di navigazione che riporta le sezioni più importanti del sito;
- Il nome dell'utente autenticato, associato alla possibilità di effettuare il logout;
- Il contenuto della pagina, provvisto dei dati richiesti dall'utente.

Tutte le view estendono **app.blade** in modo da mantenere un *look-and-feel* consistente per l'intero sito.

4.3.3 app::Http::Controllers

Le classi e le interfacce appartenenti a questa componente sono deputate alla gestione delle richieste HTTP$_G$ in entrata: ogni richiesta viene inoltrata allo specifico controller in grado di soddisfarla.

Nel framework Laravel, una *route* rappresenta l'associazione tra una richiesta HTTP$_G$ e il controller utile alla sua risoluzione, tutte le route previste sono salvate nel file `app/Http/routes.php`. Un esempio di route è il seguente:

```
Route::get('missions.textposts','WebTextPostController@index')
```

Questa route indica che la piattaforma può gestire una richiesta HTTP$_G$ inoltrata all'URL$_G$ `missions/mission_id/textposts/textpost_id` con metodo GET. La gestione è presa in carico dal metodo `index` del controller `WebTextPostController`. I segnaposto `mission_id` e `textpost_id` indicano che l'URL$_G$ prevede due parametri interi, positivi e non opzionali. Tali parametri rappresentano rispettivamente l'id del post di tipo testo che si vuole recuperare e l'id della mission alla quale il post è collegato.
Il passaggio dei parametri, contenuti nell'URL$_G$, al metodo del controller viene completamente gestito dal framework Laravel: il primo parametro dell'URL$_G$ viene passato al primo parametro del metodo, il secondo parametro dell'URL$_G$ viene passato al secondo parametro del metodo e così via. Questa peculiarità implica che la struttura di ogni controller sia influenzata dal tipo di route che deve gestire, il tipo della particolare route dipende dal numero di parametri in essa presenti.
Sono stati quindi parametrizzati diversi tipi di controller:

- **First level**: se il controller deve gestire una route con al più un parametro;
- **Second level**: se il controller deve gestire una route con al più due parametri;
- **Third level**: se il controller deve gestire una route con al più tre parametri;
- **Fourth level**: se il controller deve gestire una route con al più quattro parametri;
- **Fifth level**: se il controller deve gestire una route con al più cinque parametri.

Dall'analisi dei requisiti è emerso, inoltre, che la piattaforma deve essere accessibile sia attraverso un browser web sia attraverso le API REST esposte dalla stessa, sono stati quindi previsti due gruppi di route:

- **Route web**: utilizzate per accedere all'interfaccia grafica del CMS$_G$ da browser web;
- **Route API**: utilizzate per accedere alle API REST offerte dalla piattaforma. L'URL$_G$ per usufruire di una certa funzionalità attraverso una route API equivale all'URL$_G$ per accedere alla stessa funzionalità attraverso una route web, con l'aggiunta del prefisso `api/v1`. Ad esempio, per accedere ad un'interfaccia grafica contenente la lista di tutte le mission è possibile utilizzare l'URL$_G$ `/missions`, mentre per ottenere un pacchetto JSON contenente lo stesso risultato è possibile utilizzare l'URL$_G$ `/api/v1/missions`.

La differenza tra i controller adibiti alla gestione delle route web e i controller adibiti alla gestione delle route API consiste nella sola formattazione del risultato restituito, questo ha portato all'utilizzo del design pattern **Template Method**. Il pattern Template Method permette di definire la struttura di un algoritmo lasciando alle sottoclassi il compito di implementarne alcuni passi, in questo modo è possibile ridefinire e personalizzare parte del comportamento nelle sottoclassi evitando codice duplicato.
La struttura dell'algoritmo è solitamente definita in un metodo chiamato *template*

method, tale metodo utilizza delle *operazioni primitive* astratte che devono essere concretizzate dalle sottoclassi per specializzarne il comportamento.

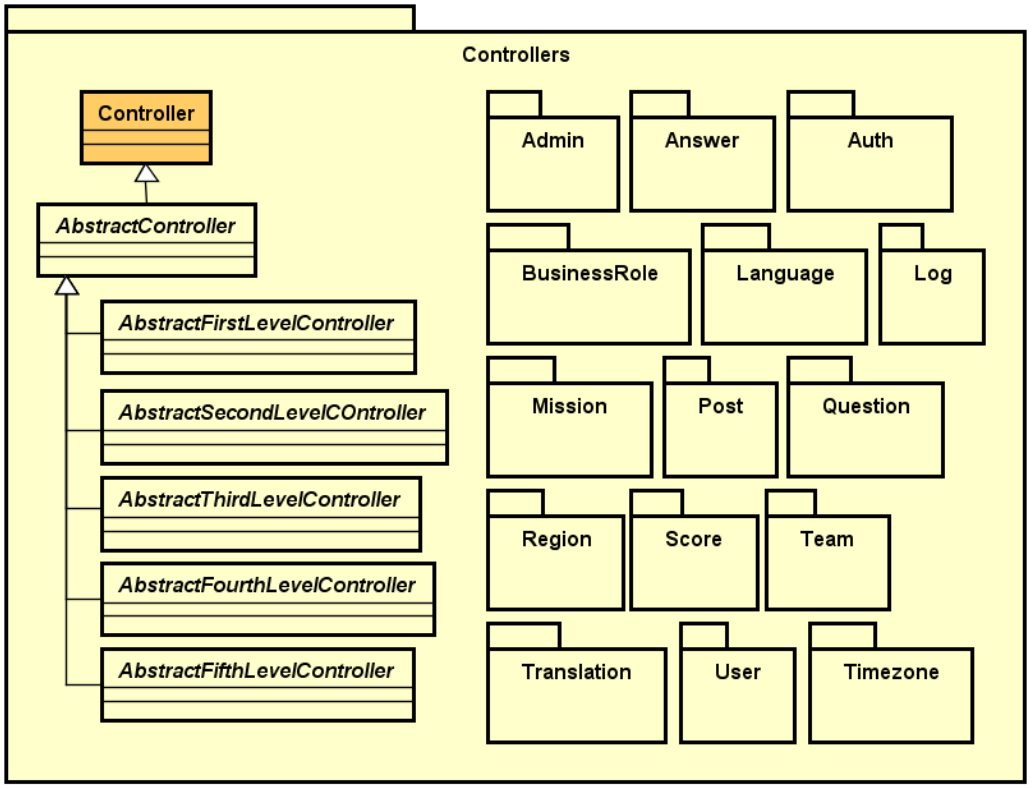


figura 4.8: Diagramma delle classi della componente Controllers

Per modellare la situazione sopra descritta sono state utilizzate delle classi astratte:

- **Controllers::AbstractController**: definisce l'attributo `repository` che ospiterà il riferimento al principale repository utilizzato dalla classe che concretizza AbstractController. Per mantenere una struttura consistente, infatti, ogni controller non può accedere direttamente al model ma deve servirsi dell'apposito repository. Questa classe espone inoltre il metodo astratto `format`, ovvero l'*operazione primitiva* utilizzata dai metodi delle sottoclassi per restituire un risultato all'utente. Questo metodo deve essere implementato dalle sottoclassi in modo da restituire i dati nel formato corretto rispetto alla modalità d'accesso dell'utente alla piattaforma;

- **Controllers::AbstractFirstLevelController**: definisce l'interfaccia dei controller di tipo *first level*;

- **Controllers::AbstractSecondLevelController**: definisce l'interfaccia dei controller di tipo *second level*;

- **Controllers::AbstractThirdLevelController**: definisce l'interfaccia dei controller di tipo *third level*;

- **Controllers::AbstractFourthLevelController**: definisce l'interfaccia dei controller di tipo *fourth level*;

- **Controllers::AbstractFifthLevelController**: definisce l'interfaccia dei controller di tipo *fifth level*.

L'interfaccia dei controller è composta da una serie di metodi utili per: creare una nuova risorsa, recuperare una determinata risorsa, aggiornare una determinata risorsa, eliminare una determinata risorsa.

Le classi appartenenti a questa componente adoperano il design pattern **Dependency Injection** per separare il loro comportamento dalla risoluzione delle dipendenze che espongono. In particolare questo pattern è utilizzato per *iniettare*, utilizzando il costruttore della classe, le dipendenze relative ai repository usati da ogni controller.

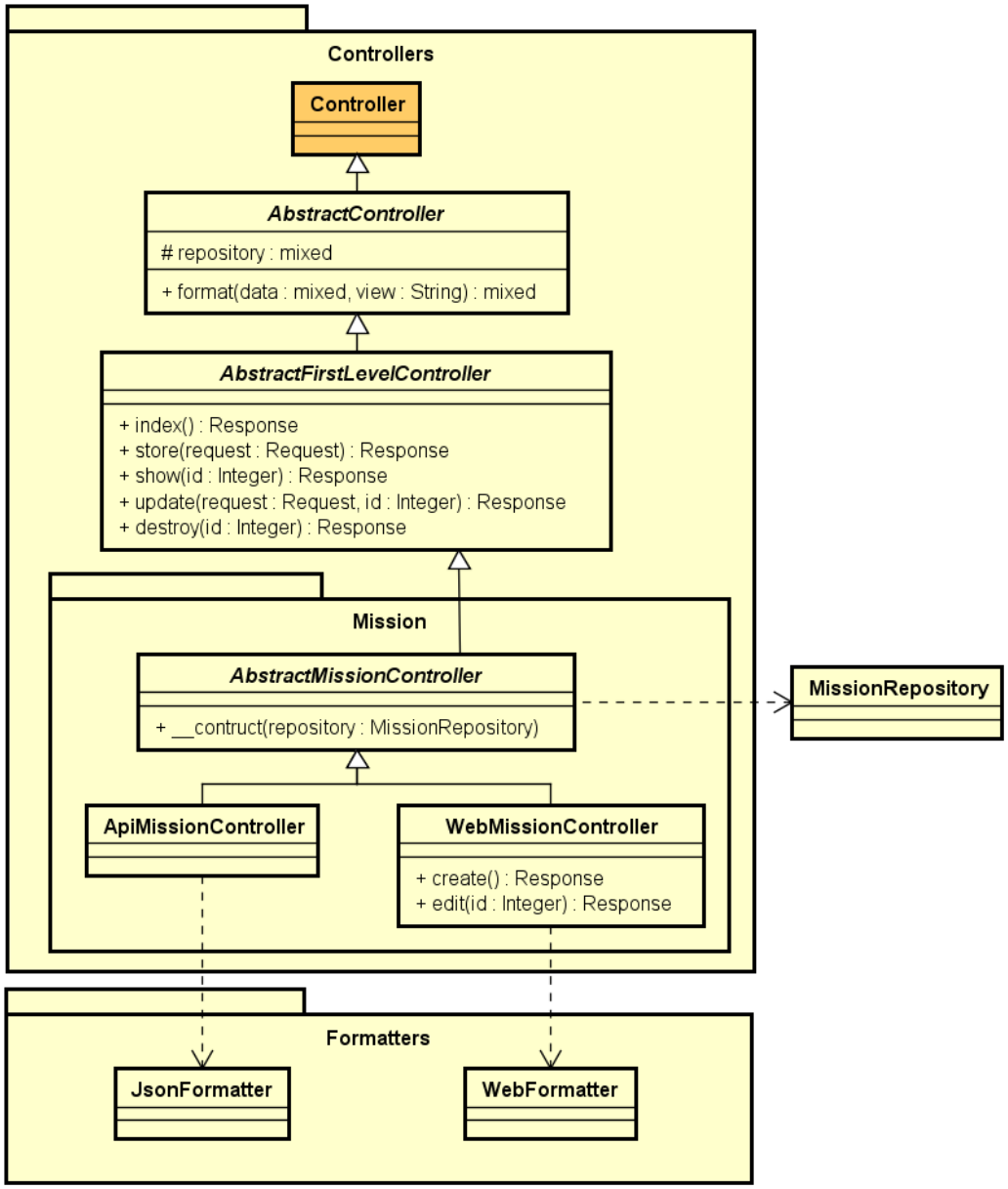


figura 4.9: Diagramma delle classi ApiMissionController e WebMissionController

Due esempi di controller di tipo *first level* sono i controller adibiti alla gestione delle mission: ApiMissionController e WebMissionController. Questi controller derivano da **AbstractMissionController** che si occcupa di implementare i metodi ereditati da AbstractFirstLevelController. Le classi concrete **ApiMissionController** e **WebMissionController** hanno invece due compiti:

- Esporre eventuali metodi specifici e relativi ad una sola modalità di accesso, ad esempio i controller che gestiscono l'accesso tramite web (come WebMissionController) espongono metodi per mostrare i form utili alla creazione e aggiornamento delle risorse, questo non è necessario per i controller che gestiscono l'accesso tramite API (come ApiMissionController);

- Implementare il metodo astratto `format`, utilizzando il costrutto appropriato della componente Formatters.

4.3.4 app::Http::Formatters

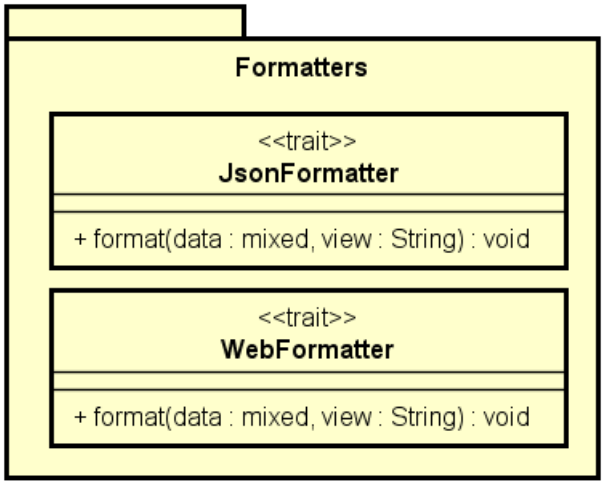


figura 4.10: Diagramma delle classi della componente Formatters

La differenza fondamentale tra l'accesso tramite browser web e l'accesso tramite API REST riguarda la formattazione dei dati ottenuti come risultato della richiesta:

- Se l'accesso avviene tramite browser web, l'utente si aspetta di poter interagire con un'interfaccia grafica;
- Se l'accesso avviene tramite API REST, l'utente si aspetta di poter interagire con un formato di dati strutturato. In questo caso è stato scelto di fornire all'utente un pacchetto di dati in formato JSON$_G$ in quando tale formato è oggi largamente utilizzato nello scambio di dati sul web, inoltre il framework Laravel ne rende molto agevole la manipolazione.

Questa componente contiene due Trait$_G$ che si occupano proprio di formattare il risultato della richiesta in modo adeguato:

- **Formatters::JsonFormatter**: restituisce i dati in formato JSON$_G$;
- **Formatters::WebFormatter**: restituisce i dati attraverso una web view.

Entrambi i Trait$_G$ espongono un unico metodo: `format`. Essi vanno intesi come dei pacchetti di codice pre-confezionato che agevolano lo sviluppatore nell'implementazione del metodo `format` richiesto per la realizzazione di un controller concreto. In questo modo la scrittura di codice duplicato viene ridotta al minimo: lo sviluppatore deve solamente scegliere quale formatter utilizzare e non deve preoccuparsi di scrivere la procedura per la formattazione, in quanto questa è già incapsulata nel formatter scelto.

Appendice A

Casi d'uso secondari

A.1 Caso d'uso UC3: gestione user

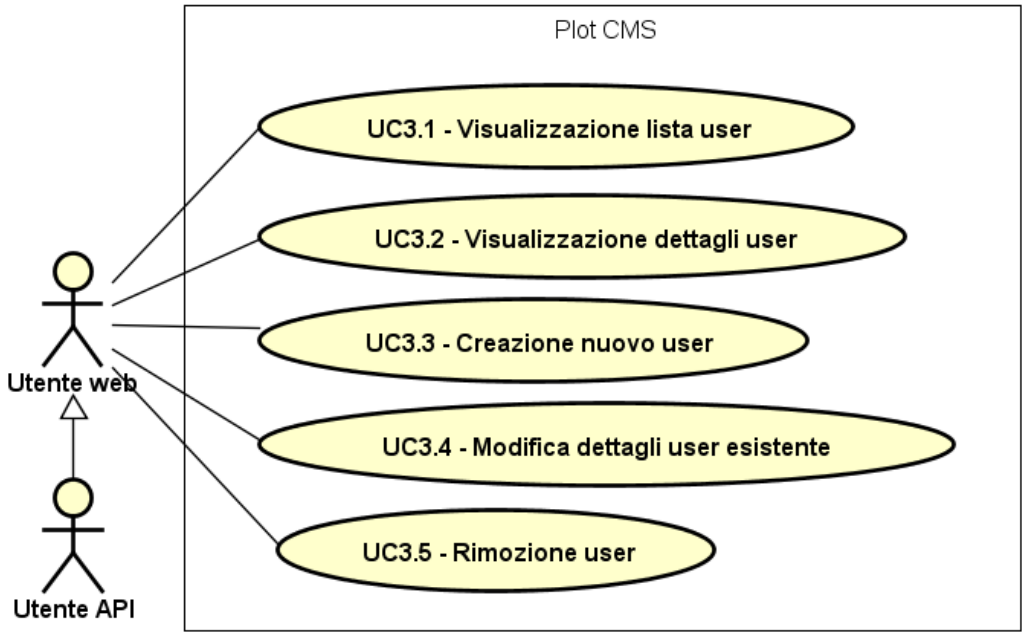


figura A.1: Caso d'uso UC3: gestione user

- **Attori**: utente web, utente API;
- **Descrizione**: un utente autenticato deve poter gestire gli aspetti di base legati agli utenti del gioco: creazione, visualizzazione, modifica e rimozione;
- **Precondizione**: l'utente accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema;
- **Flusso principale degli eventi**:
    1. L'utente autenticato può visualizzare la lista degli utenti del gioco;

2. L'utente autenticato può visualizzare i dettagli di uno specifico utente del gioco (UC3.2);
3. L'utente autenticato può creare un nuovo utente del gioco (UC3.3);
4. L'utente autenticato può modificare i dettagli di uno specifico utente di gioco (UC3.4);
5. L'utente autenticato può rimuovere un utente del gioco.

- **Postcondizione**: il sistema ha erogato le funzionalità richieste dall'utente.

A.2 Caso d'uso UC3.2: visualizzazione dettagli user

- **Attori**: utente web, utente API;
- **Descrizione**: l'utente autenticato deve poter visualizzare i dettagli relativi ad uno specifico utente del gioco;
- **Precondizione**: l'utente accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema. L'utente del gioco che si vuole visualizzare deve esistere;
- **Flusso principale degli eventi**:
  1. L'utente autenticato visualizza il nome dell'utente di gioco;
  2. L'utente autenticato visualizza il cognome dell'utente di gioco;
  3. L'utente autenticato visualizza l'email dell'utente di gioco;
  4. L'utente autenticato visualizza l'ambito lavorativo dell'utente di gioco;
  5. L'utente autenticato visualizza la regione di appartenenza dell'utente di gioco;
  6. L'utente autenticato visualizza la lingua preferita dell'utente di gioco;
  7. L'utente autenticato visualizza il team dell'utente di gioco.
- **Postcondizione**: il sistema mostra all'utente i dettagli dell'utente del gioco richiesto.

A.3 Caso d'uso UC3.3: creazione nuovo user

- **Attori**: utente web, utente API;
- **Descrizione**: l'utente autenticato deve poter creare un nuovo utente del gioco;
- **Precondizione**: l'utente accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema;
- **Flusso principale degli eventi**:
  1. L'utente autenticato inserisce il nome dell'utente del gioco;
  2. L'utente autenticato inserisce il cognome dell'utente del gioco;

3. L'utente autenticato inserisce l'indirizzo email dell'utente del gioco;
4. L'utente autenticato può inserire la password dell'utente del gioco;
5. L'utente autenticato può inserire l'ambito lavorativo dell'utente del gioco;
6. L'utente autenticato può inserire la regione d'appartenenza dell'utente del gioco;
7. L'utente autenticato può inserire la lingua preferita dell'utente del gioco;
8. L'utente autenticato può inserire il team dell'utente del gioco.

- **Postcondizione**: il sistema crea un nuovo utente del gioco e ne mostra i dettagli all'utente.

A.4 Caso d'uso UC3.4: modifica dettagli user esistente

- **Attori**: utente web, utente API;
- **Descrizione**: l'utente autenticato deve poter modificare i dettagli relativi ad un utente del gioco esistente;
- **Precondizione**: l'utente accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema. L'utente del gioco che si vuole modificare deve esistere;
- **Flusso principale degli eventi**:
  1. L'utente autenticato può modificare il nome dell'utente del gioco;
  2. L'utente autenticato può modificare il cognome dell'utente del gioco;
  3. L'utente autenticato può modificare l'indirizzo email dell'utente del gioco;
  4. L'utente autenticato può modificare l'ambito lavorativo dell'utente del gioco;
  5. L'utente autenticato può modificare la regione di appartenenza dell'utente del gioco;
  6. L'utente autenticato può modificare la lingua preferita dell'utente del gioco;
  7. L'utente autenticato può modificare il team di appartenenza dell'utente del gioco;
  8. L'utente autenticato può modificare il team di appartenenza dell'utente del gioco;
  9. L'utente autenticato può modificare il team di appartenenza dell'utente del gioco.
- **Postcondizione**: il sistema apporta le modifiche richieste e mostra i dettagli dell'utente di gioco modificato all'utente autenticato.

A.5 Caso d'uso UC4: gestione team

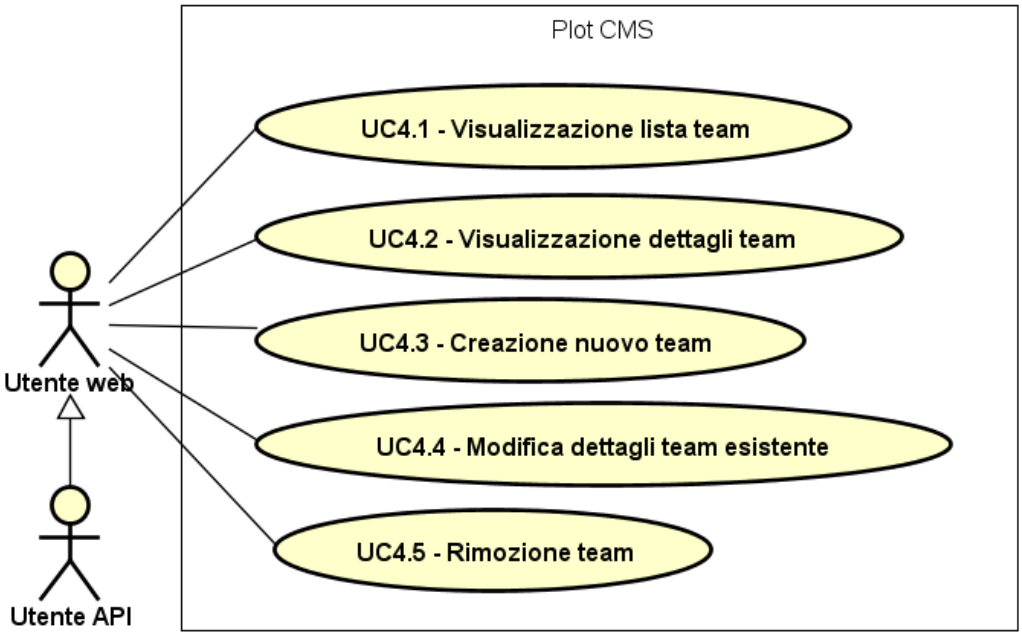


figura A.2: Caso d'uso UC4: gestione team

- **Attori**: utente web, utente API;
- **Descrizione**: un utente autenticato deve poter gestire gli aspetti di base legati ai team: creazione, visualizzazione, modifica e rimozione;
- **Precondizione**: l'utente accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema;
- **Flusso principale degli eventi**:
  1. L'utente autenticato può visualizzare la lista dei team (UC4.1);
  2. L'utente autenticato può visualizzare i dettagli di uno specifico team (UC4.2);
  3. L'utente autenticato può creare un team (UC4.3);
  4. L'utente autenticato può modificare i dettagli di uno specifico team (UC4.4);
  5. L'utente autenticato può rimuovere un team (UC4.5).
- **Postcondizione**: il sistema ha erogato le funzionalità richieste dall'utente.

A.6 Caso d'uso UC4.1: visualizzazione lista team

- **Attori**: utente web, utente API;

- **Descrizione**: l'utente autenticato deve poter visualizzare una lista di tutti i team in gioco. Ogni team deve riportare il numero di componenti e il punteggio totale, ovvero la somma dei punteggi ottenuti nelle mission disputate;
- **Precondizione**: l'utente autenticato accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema;
- **Flusso principale degli eventi**:
  1. L'utente autenticato visualizza la lista dei team in gioco.
- **Postcondizione**: il sistema mostra la lista dei team in gioco.

A.7 Caso d'uso UC4.2: visualizzazione dettagli team

- **Attori**: utente web, utente API;
- **Descrizione**: l'utente autenticato deve poter visualizzare i dettagli relativi ad uno specifico team in gioco;
- **Precondizione**: l'utente accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema. Il team che si vuole visualizzare deve esistere;
- **Flusso principale degli eventi**:
  1. L'utente autenticato visualizza il nome del team;
  2. L'utente autenticato visualizza l'URL dell'immagine rappresentativa del team;
  3. L'utente autenticato visualizza la lista degli utenti del gioco appartenenti al team.
- **Postcondizione**: il sistema mostra all'utente i dettagli del team richiesto.

A.8 Caso d'uso UC4.3: creazione nuovo team

- **Attori**: utente web, utente API;
- **Descrizione**: l'utente autenticato deve poter creare un nuovo team di gioco;
- **Precondizione**: l'utente accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema;
- **Flusso principale degli eventi**:
  1. L'utente autenticato inserisce il nome del team;
  2. L'utente autenticato può inserire l'URL di un'immagine rappresentativa del team.
- **Postcondizione**: il sistema crea un nuovo team e ne mostra i dettagli all'utente.

A.9 Caso d'uso UC4.4: modifica dettagli team esistente

- **Attori**: utente web, utente API;
- **Descrizione**: l'utente autenticato deve poter modificare i dettagli relativi ad un team esistente;
- **Precondizione**: l'utente accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema. Il team che si vuole modificare deve esistere;
- **Flusso principale degli eventi**:
  1. L'utente autenticato può modificare il nome del team;
  2. L'utente autenticato può modificare l'URL dell'immagine rappresentativa del team.
- **Postcondizione**: il sistema apporta le modifiche richieste e mostra i dettagli del team modificato all'utente.

A.10 Caso d'uso UC4.5: rimozione team

- **Attori**: utente web, utente API;
- **Descrizione**: l'utente autenticato deve poter rimuovere un team di gioco;
- **Precondizione**: l'utente autenticato accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema. Il team che si vuole rimuovere deve esistere;
- **Flusso principale degli eventi**:
  1. L'utente autenticato rimuove un particolare team di gioco .
- **Postcondizione**: Il team viene rimosso dal sistema e non può più essere recuperato. Il sistema mostra all'utente una lista degi team ancora presenti nel sistema.

A.11 Caso d'uso UC5: gestione admin

- **Attori**: utente web, utente API;
- **Descrizione**: un utente autenticato deve poter gestire gli aspetti di base legati agli amministratori della piattaforma: creazione, visualizzazione, modifica e rimozione;
- **Precondizione**: l'utente accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema;
- **Flusso principale degli eventi**:

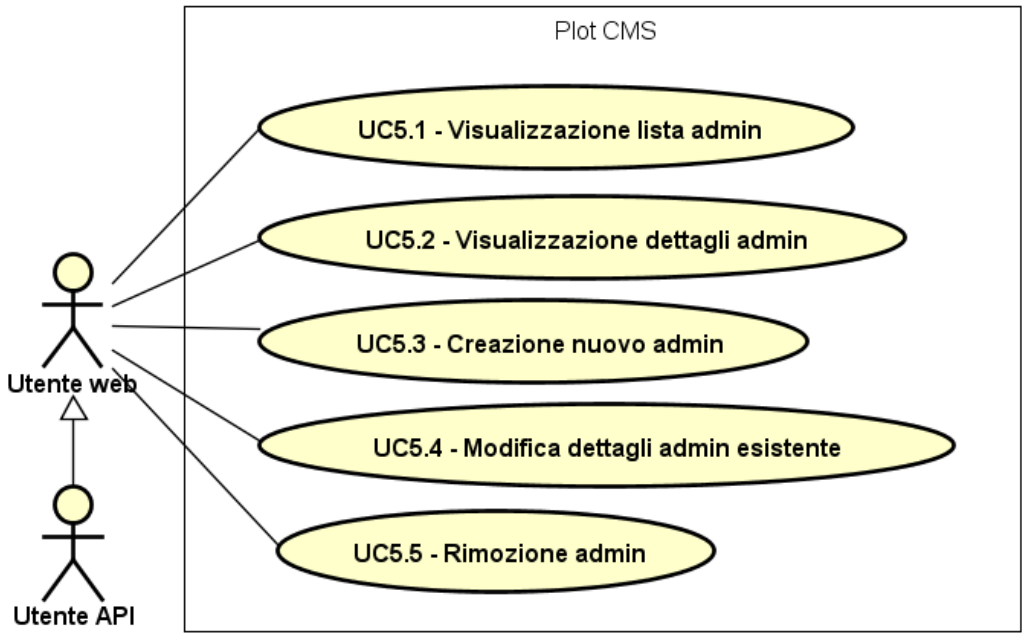


figura A.3: Caso d'uso UC5: gestione admin

1. L'utente autenticato può visualizzare la lista degli amministratori della piattaforma (UC5.1);
2. L'utente autenticato può visualizzare i dettagli di uno specifico amministratore della piattaforma (UC5.2);
3. L'utente autenticato può creare un nuovo amministratore della piattaforma (UC5.3);
4. L'utente autenticato può modificare i dettagli di uno specifico amministratore della piattaforma (UC5.4);
5. L'utente autenticato può rimuovere un amministratore della piattaforma (UC5.5).

- **Postcondizione**: il sistema ha erogato le funzionalità richieste dall'utente.

A.12 Caso d'uso UC5.1: visualizzazione lista admin

- **Attori**: utente web, utente API;
- **Descrizione**: l'utente autenticato deve poter visualizzare una lista di tutti gli amministratori della piattaforma. Gli amministratori sono coloro che possono utilizzare le funzionalità offerte dalla piattaforma;
- **Precondizione**: l'utente autenticato accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema;
- **Flusso principale degli eventi**:

1. L'utente autenticato visualizza la lista degli amministratori della piattaforma.

- **Postcondizione**: il sistema mostra la lista degli amministratori della piattaforma.

A.13 Caso d'uso UC5.2: visualizzazione dettagli admin

- **Attori**: utente web, utente API;
- **Descrizione**: l'utente autenticato deve poter visualizzare i dettagli relativi ad uno specifico amministratore della piattaforma;
- **Precondizione**: l'utente accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema. L'amministratore che si vuole visualizzare deve esistere;
- **Flusso principale degli eventi**:
  1. L'utente autenticato visualizza il nome dell'amministratore della piattaforma;
  2. L'utente autenticato visualizza il cognome dell'amministratore della piattaforma;
  3. L'utente autenticato visualizza l'email dell'amministratore della piattaforma.
- **Postcondizione**: il sistema mostra all'utente i dettagli dell'amministratore richiesto.

A.14 Caso d'uso UC5.3: creazione nuovo admin

- **Attori**: utente web, utente API;
- **Descrizione**: l'utente autenticato deve poter creare un nuovo amministratore della piattaforma ;
- **Precondizione**: l'utente accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema;
- **Flusso principale degli eventi**:
  1. L'utente autenticato inserisce il nome dell'amministratore della piattaforma;
  2. L'utente autenticato inserisce il cognome dell'amministratore della piattaforma;
  3. L'utente autenticato inserisce l'email dell'amministratore della piattaforma;
  4. L'utente autenticato inserisce la password dell'amministratore della piattaforma.

- **Postcondizione**: il sistema crea un nuovo amministratore della piattaforma e ne mostra i dettagli all'utente.

A.15 Caso d'uso UC5.4: modifica dettagli admin esistente

- **Attori**: utente web, utente API;
- **Descrizione**: l'utente autenticato deve poter modificare i dettagli relativi ad un amministratore esistente;
- **Precondizione**: l'utente accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema. L'amministratore che si vuole modificare deve esistere;
- **Flusso principale degli eventi**:
  1. L'utente autenticato può modificare il nome dell'amministratore;
  2. L'utente autenticato può modificare il cognome dell'amministratore;
  3. L'utente autenticato può modificare l'email dell'amministratore.
- **Postcondizione**: il sistema apporta le modifiche richieste e mostra i dettagli dell'amministratore modificato all'utente.

A.16 Caso d'uso UC5.5: rimozione admin

- **Attori**: utente web, utente API;
- **Descrizione**: l'utente autenticato deve poter rimuovere un amministratore della piattaforma;
- **Precondizione**: l'utente autenticato accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema. L'amministratore che si vuole rimuovere deve esistere;
- **Flusso principale degli eventi**:
  1. L'utente autenticato rimuove un particolare amministratore della piattaforma.
- **Postcondizione**: l'amministratore viene rimosso dal sistema e non può più essere recuperato. Il sistema mostra all'utente una lista degli amministratori ancora presenti nel sistema.

A.17 Caso d'uso UC6: gestione log

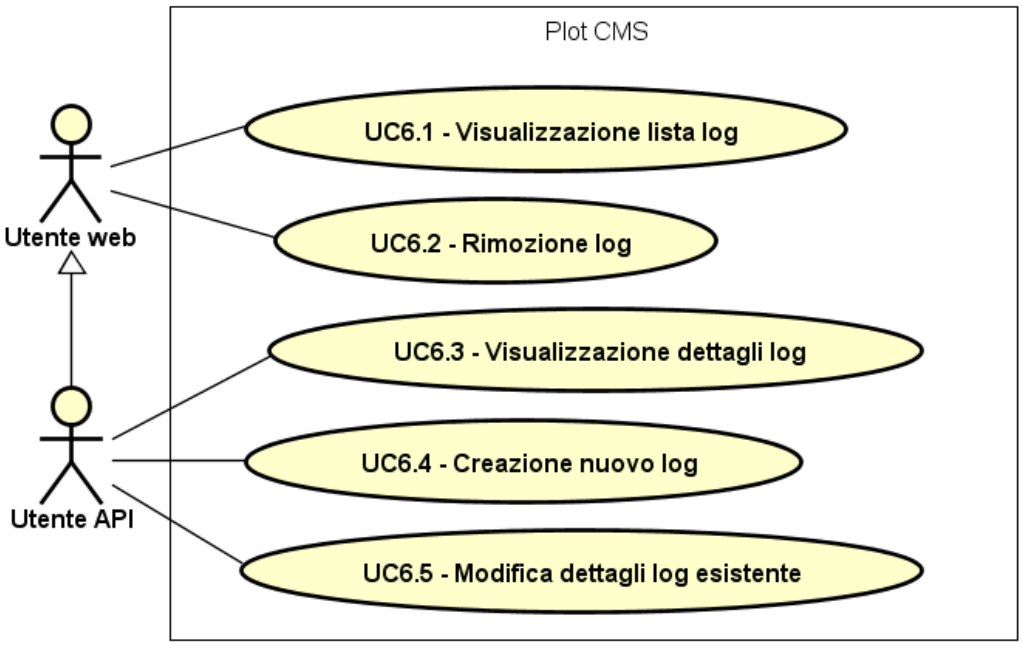


figura A.4: Caso d'uso UC6: gestione log

- **Attori**: utente web, utente API;
- **Descrizione**: un utente autenticato che accede alla piattaforma dal web deve poter gestire la visualizzazione e la rimozione dei log generati dagli utenti del gioco. In aggiunta a ciò, un utente autenticato che accede alla piattaforma attraverso le API deve poter creare un nuovo log e modificare un log esistente;
- **Precondizione**: l'utente accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema;
- **Flusso principale degli eventi**:
  1. L'utente autenticato che accede tramite web può visualizzare la lista dei log (UC6.1);
  2. L'utente autenticato che accede tramite web può rimuovere un log (UC6.2);
  3. L'utente autenticato che accede tramite API può visualizzare i dettagli di uno specifico log (UC6.3);
  4. L'utente autenticato che accede tramite API può creare un nuovo log (UC6.4);
  5. L'utente autenticato che accede tramite API può modificare i dettagli di uno specifico log (UC6.5).
- **Postcondizione**: il sistema ha erogato le funzionalità richieste dall'utente.

A.18 Caso d'uso UC6.1: visualizzazione lista log

- **Attori**: utente web, utente API;
- **Descrizione**: l'utente autenticato deve poter visualizzare una lista di tutti i log generati dalle richieste HTTP effettuate dagli utenti di gioco ;
- **Precondizione**: l'utente autenticato accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema;
- **Flusso principale degli eventi**:
  1. L'utente autenticato visualizza la lista dei log generati dalle richieste HTTP effettuate dagli utenti di gioco .
- **Postcondizione**: il sistema mostra la lista dei log generati dalle richieste HTTP effettuate dagli utenti di gioco .

A.19 Caso d'uso UC6.2: rimozione log

- **Attori**: utente web, utente API;
- **Descrizione**: l'utente autenticato deve poter rimuovere un log generato da una richiesta HTTP effettuata da un utente di gioco;
- **Precondizione**: l'utente autenticato accede alla piattaforma attraverso un browser web o attraverso le API REST esposte dal sistema. Il log che si vuole rimuovere deve esistere;
- **Flusso principale degli eventi**:
  1. L'utente autenticato rimuove un log generato da una richiesta HTTP effettuata da un utente di gioco.
- **Postcondizione**: il log viene rimosso dal sistema e non può più essere recuperato. Il sistema mostra all'utente una lista dei log ancora presenti nel sistema.

A.20 Caso d'uso UC6.3: visualizzazione dettagli log

- **Attori**: utente API;
- **Descrizione**: l'utente autenticato deve poter visualizzare i dettagli di un log generato da una richiesta HTTP effettuata da un utente di gioco;
- **Precondizione**: l'utente autenticato accede alla piattaforma attraverso le API REST esposte dal sistema. Il log che si vuole visualizzare deve esistere;
- **Flusso principale degli eventi**:
  1. L'utente autenticato visualizza l'utente di gioco che ha effettuato la richiesta HTTP;

2. L'utente autenticato visualizza il metodo HTTP utilizzato per la richiesta;
3. L'utente autenticato visualizza l'URL alla quale è stata inoltrata la richiesta HTTP;
4. L'utente autenticato visualizza lo user agent utilizzato per inoltrare la richiesta HTTP;
5. L'utente autenticato visualizza l'indirizzo IP che ha effettuato la richiesta HTTP;
6. L'utente autenticato visualizza i dati aggiuntivi trasportati dalla richiesta HTTP;
7. L'utente autenticato visualizza il timestamp relativo alla creazione del log.

- **Postcondizione**: i dettagli del log vengono mostrati all'utente.

A.21 Caso d'uso UC6.4: creazione nuovo log

- **Attori**: utente API;
- **Descrizione**: l'utente autenticato deve poter creare un nuovo log utilizzando i dettagli di da una richiesta HTTP effettuata da un utente di gioco;
- **Precondizione**: l'utente autenticato accede alla piattaforma attraverso le API REST esposte dal sistema. Il log che si vuole visualizzare deve esistere;
- **Flusso principale degli eventi**:
  1. L'utente autenticato inserisce l'user che ha effettuato la richiesta HTTP;
  2. L'utente autenticato inserisce il metodo HTTP utilizzato per la richiesta;
  3. L'utente autenticato inserisce l'URL indirizzato dalla richiesta HTTP;
  4. L'utente autenticato inserisce lo user agent che ha effettuato la richiesta HTTP;
  5. L'utente autenticato inserisce l'indirizzo IP che ha effettuato la richiesta HTTP;
  6. L'utente autenticato inserisce i dati aggiuntivi trasportati dalla la richiesta HTTP;
  7. L'utente autenticato inserisce il timestamp relativo alla creazione del corrente log.
- **Postcondizione**: Il sistema crea un nuovo log e ne mostra i dettagli all'utente.

A.22 Caso d'uso UC6.5: modifica dettagli log esistente

- **Attori**: utente API;
- **Descrizione**: l'utente autenticato deve poter modificare log esistente ;
- **Precondizione**: l'utente autenticato accede alla piattaforma attraverso le API REST esposte dal sistema. Il log che si vuole modificare deve esistere;

- **Flusso principale degli eventi**:
  1. L'utente autenticato può modificare l'user che ha effettuato la richiesta HTTP;
  2. L'utente autenticato può modificare il metodo HTTP utilizzato per la richiesta;
  3. L'utente autenticato può modificare l'URL indirizzato dalla richiesta HTTP;
  4. L'utente autenticato può modificare lo user agent che ha effettuato la richiesta HTTP;
  5. L'utente autenticato può modificare l'indirizzo IP che ha effettuato la richiesta HTTP;
  6. L'utente autenticato può modificare i dati aggiuntivi trasportati dalla la richiesta HTTP;
  7. L'utente autenticato può modificare il timestamp relativo alla creazione del corrente log;
  8. L'utente autenticato può modificare il timestamp relativo alla creazione del corrente log;
  9. L'utente autenticato può modificare il timestamp relativo alla creazione del corrente log.
- **Postcondizione**: Il sistema apporta le modifiche richieste e mostra i dettagli del log modificato all'utente.

Appendice B

API REST

Negli anni 2000, con la diffusione crescente del web, inizia a prendere piede il concetto di **web service**: sistemi software che supportano l'interazione tra applicazioni diverse utilizzando gli standard web. Il meccanismo alla base dei web service permette di far comunicare applicazioni sviluppate con linguaggi di programmazione differenti e residenti su macchine differenti. In questo modo diventa possibile creare architetture modulari che esternalizzino le proprie funzionalità, in modo da ottenere un sistema facilmente componibile.

Ad oggi esistono fondamentalmente due approcci alla creazione di web service: **SOAP** e **REST**, il secondo risulta essere più diffuso in quanto permette la creazione di servizi altamente efficienti e facilmente scalabili.

B.1 REST

L'approccio *REST*, acronimo di "*REpresentational State Transfer*", si ispira ai principi architetturali tipici del web e si basa sul concetto di *risorsa*.
Secondo l'approccio REST:

- Deve essere possibile identificare univocamente una risorsa;
- I metodi forniti dal protocollo HTTP$_G$ devono essere usati in modo esplicito;
- Ogni risorsa deve essere autodescrittiva;
- Deve essere possibile collegare tra loro più risorse;
- La comunicazione deve avvenire senza stato.

B.1.1 Identificazione univoca delle risorse

Una **risorsa** è un qualsiasi elemento oggetto di elaborazione, solitamente mantenuto in un sistema di persistenza dei dati. L'identificazione si concretizza nell'utilizzo dello schema **URI**: ogni risorsa (sia essa un indirizzo web, un documento, un'immagine, ecc.) può essere identificata univocamente da una stringa.

Seguendo questa convenzione è possibile, ad esempio, identificare uno specifico utente attraverso la stringa:

```
http://www.myapplication.com/users/1
```

Come principio generale conviene evitare di usare verbi negli URI, infatti risulta più appropriato utilizzare nomi in quanto gli URI identificano risorse e non azioni.

B.1.2 Uso esplicito dei metodi HTTP

Il protocollo HTTP$_G$ mette a disposizione nativamente i metodi (o verbi): GET, POST, PUT, DELETE. Questi metodi possono essere mappati sulle operazioni CRUD$_G$:

HTTP	CRUD	Descrizione
POST	Create	Crea una nuova risorsa
GET	Read	Recupera una risorsa esistente
PUT	Update	Aggiorna una risorsa
DELETE	Delete	Elimina una risorsa

tabella B.1: Associazione tra metodi HTTP e operazioni CRUD

Seguendo questa convenzione è possibile, ad esempio, creare un nuovo utente inoltrando una richiesta HTTP$_G$ con metodo POST all'URL$_G$:

```
http://www.myapplication.com/users
```

B.1.3 Risorse autodescrittive

Le risorse sono concettualmente diverse dalla loro rappresentazione che viene restituita al client. Questo significa che i web service non inviano ai client direttamente un record del database, ma delle rappresentazioni strutturate secondo la richiesta. Il tipo di rappresentazione viene allegato dal server alla risposta che fornisce al client e può essere, ad esempio, il formato JSON$_G$.

B.1.4 Collegamenti tra risorse

Le risorse che sono in relazione tra di loro devono esplicitare questo collegamento attraverso link ipertestuali. Questo fa in modo che tutta la conoscenza che il client deve avere sulla risorsa sia presente nella sua rappresentazione o sia accessibile tramite collegamenti ipertestuali.

B.1.5 Comunicazione senza stato

Il principio della *comunicazione stateless* è una delle caratteristiche fondamentali del protocollo HTTP$_G$: ogni richiesta non ha alcuna relazione con le richieste future o passate. Questo significa che la gestione dello stato dell'applicazione deve essere appannaggio del client, non del server.

Glossario

Alternate Reality Game spesso riferito anche con l'acronimo "ARG", è una situazione di gioco che collega internet al mondo reale. Solitamente si sviluppa attraverso numerosi strumenti web (blog, email, mini-siti) e presenta al giocatore una storia misteriosa con indizi che puntano al mondo reale (per esempio a monumenti o a veri e propri oggetti nascosti in determinate località). 5

Back end termine che denota l'insieme delle applicazioni, relative ad una piattaforma web, con le quali l'utente non interagisce direttamente ma che sono essenziali al funzionamento del sistema. 10

B2B acronimo di "*business-to-business*". Locuzione utilizzata per descrivere le transazioni commerciali elettroniche tra imprese, distinguendole da quelle che intercorrono tra le imprese e altri gruppi, come quelle tra una ditta e i clienti individuali (B2C, acronimo di "*business-to-consumer*") oppure quelle tra una impresa e il governo (B2G, acronimo di "*business-to-government*"). 2

CMS acronimo di "*Content Management System*". Con questo termine si indica uno strumento software, installato su un server web, il cui obiettivo è facilitare la gestione dei contenuti di siti web, soprattutto a coloro che non hanno conoscenze specifiche sulla programmazione web. 9, 11–13, 17, 58, 63, 65

CRUD acronimo di "*Create, Read, Update, Delete*". Riassume le operazioni di base che possono essere effettuate su un sistema di persistenza dati: creazione di una risorsa, visualizzazione di una risorsa, aggiornamento di una risorsa e rimozione di una risorsa. 58, 86

Finanziamento early stage investimento atto a sostenere le fasi iniziali di un'azienda o di un business. 2

Front end termine che denota la parte, di una piattaforma web, visibile all'utente e con la quale esso può interagire. Ci si riferisce a questo concetto anche con il termine "*interfaccia utente*". 10–12

HTTP acronimo di "*HyperText Transfer Protocol*". Protocollo a livello applicativo usato come principale sistema per la trasmissione d'informazioni sul web. 11, 18, 64, 65, 85, 86

International School prevede un percorso didattico che si articola seguendo il modello anglosassone ed è certificato dall' *International Baccalaureate Organization*. Al termine del percorso, con il conseguimento dell'*IB-Diploma Programme* gli studenti ottengono un diploma equipollente alla Maturità italiana con un anno di anticipo rispetto al percorso italiano. Hanno accesso a tutte le Università italiane e ai più prestigiosi atenei del mondo. 3

IoT acronimo di "*internet of things*". Neologismo riferito ad una evoluzione dell'uso della rete: gli oggetti si rendono riconoscibili e reagiscono di conseguenza grazie al fatto di poter comunicare dati su se stessi e accedere ad informazioni aggregate da parte di altri. Le sveglie suonano prima in caso di traffico, le scarpe da ginnastica trasmettono tempi, velocità e distanza per gareggiare in tempo reale con persone dall'altra parte del globo, i contenitori delle medicine avvisano i familiari se si dimentica di prendere il farmaco. Tutti gli oggetti possono acquisire un ruolo attivo grazie al collegamento alla Rete. 2

JSON acronimo di "*JavaScript Object Notation*", è un formato adatto all'interscambio di dati fra applicazioni client-server. 69, 86

Mentor nell'ambito del mentoring, è il soggetto con più esperienza. Deve avere capacità relazionali, saper condurre colloqui e porre domande sagge, deve saper gestire le fasi del processo di mentoring. 2

Mentoring metodologia di formazione che fa riferimento a una relazione uno a uno tra un soggetto con più esperienza (mentor) e uno con meno esperienza (*junior*, *mentee*, *protégé*), cioè un allievo, al fine di far sviluppare a quest'ultimo competenze in ambito formativo, lavorativo e sociale e di sviluppare autostima, a livello educativo-scolastico. 2

MySQL RDBMS open source, rappresenta una delle tecnologie più diffuse nel mondo dell'IT. Nasce nel 1996 per opera dell'azienda svedese Tcx e viene distribuito liberamente per favorirne la crescita. 10, 12, 51

ORM acronimo di "*Object-Relational Mapping*". Con questo termine ci si riferisce ad una tecnica di programmazione che favorisce l'integrazione tra sistemi software orientati agli oggetti e sistemi . Più nello specifico, un prodotto ORM fornisce accesso alla persistenza dei dati attraverso un'interfaccia ad oggetti, astraendo così dai dettagli dello specifico RDBMS utilizzato. 13, 57, 59, 61

RDBMS acronimo di "*Relational Database Management System*". Con questo termine ci si riferisce ad un sistema per la gestione di basi di dati relazionali. 88

Ridondanza nell'ambito dei database, un dato è ridondante se può essere derivato da altri dati. 51

Snake case la notazione snake case è la pratica di scrivere parole composte o frasi separandone le parti tramite un trattino basso (_), solitamente lasciandone tutti i caratteri in minuscolo. 20

Startup in economia, con questo termine, si indica una nuova impresa nelle forme di un'organizzazione temporanea o una società di capitali in cerca di un business model ripetibile e scalabile. La scalabilità è un elemento cardine di questa tipologia di impresa. L'avvio di un'attività imprenditoriale non scalabile, come l'apertura di un ristorante, non coincide dunque con la creazione di una startup ma, piuttosto, di una società tradizionale. 1–3

Trait costrutto del linguaggio PHP. I *Trait* sono pensati per ridurre alcuni limiti imposti dall'assenza della derivazione multipla in PHP, permettendo allo sviluppatore di riutilizzare un insieme di metodi. Un Trait è simile ad una classe ma deve essere usato solo per raggruppare delle funzionalità, non è consentito istanziare un Trait al di fuori di una classe . 69

URL acronimo di "*Uniform Resource Locator*". Nella terminologia delle telecomunicazioni e dell'informatica, è una sequenza di caratteri che identifica univocamente l'indirizzo di una risorsa in Internet, tipicamente presente su un host server, come ad esempio un documento, un'immagine, un video, rendendola accessibile ad un client che ne faccia richiesta attraverso l'utilizzo di un web browser. 18, 23, 24, 65, 86

Bibliografia

Riferimenti bibliografici

Erich Gamma Ralph Johnson, Richard Helm e John Vlissides. *Design Patterns*. Addison-Wesley, 1994.

Paolo Atzeni Stefano Ceri, Stefano Paraboschi e Riccardo Torlone. *Basi di Dati*. McGraw Hill, 2014.

Siti Web consultati

Laravel Documentation. URL: https://laravel.com/docs/ (cit. a p. 61).