



ALMA MATER STUDIORUM  
UNIVERSITÀ DI BOLOGNA

ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA  
CAMPUS DI CESENA

---

DIPARTIMENTO DI INFORMATICA – SCIENZA E INGEGNERIA  
Corso di Laurea in Ingegneria e Scienze Informatiche

# PROGETTAZIONE E SVILUPPO DI UNA PIATTAFORMA DIGITALE PER IL PEER TUTORING: IL PROTOTIPO GOAT

*Elaborato in*  
SISTEMI EMBEDDED E INTERNET OF THINGS

*Relatore*

Prof. ALESSANDRO RICCI

*Corelatore*

Prof.ssa MARTINA GIUNCHI

*Presentata da*

SOHAIL MAMA

Anno Accademico 2026 – 2027



*Dedico questa tesi ai figli di seconda generazione che hanno saputo far fiorire il futuro che sognavano, ma soprattutto ai loro genitori, che hanno lasciato terra, certezze e sé stessi pur di donar loro un domani migliore. A voi, che avete trasformato sacrifici in possibilità e silenzi in speranza: ogni conquista dei vostri figli porta la vostra impronta.*



# Indice

|                                                                          |            |
|--------------------------------------------------------------------------|------------|
| <b>Introduzione</b>                                                      | <b>vii</b> |
| <b>1 Progetto GOAT: Analisi del contesto e dei requisiti</b>             | <b>1</b>   |
| 1.1 Contesto applicativo . . . . .                                       | 1          |
| 1.2 Analisi del problema . . . . .                                       | 2          |
| 1.3 Attori del sistema e logica dei ruoli . . . . .                      | 3          |
| 1.4 Primo mockup del sistema . . . . .                                   | 4          |
| 1.5 Raccolta dei requisiti e analisi . . . . .                           | 6          |
| 1.5.1 Diagramma dei casi d'uso . . . . .                                 | 6          |
| 1.5.2 Diagramma di sequenza . . . . .                                    | 7          |
| 1.6 Analisi del dominio . . . . .                                        | 10         |
| 1.7 Analisi dei requisiti . . . . .                                      | 11         |
| <b>2 Progettazione e implementazione del sistema GOAT</b>                | <b>15</b>  |
| 2.1 Dal modello concettuale all'implementazione . . . . .                | 15         |
| 2.2 Architettura del sistema . . . . .                                   | 15         |
| 2.2.1 Diagramma dei componenti . . . . .                                 | 16         |
| 2.2.2 Diagramma delle classi . . . . .                                   | 17         |
| 2.3 Gestione dell'autenticazione e dello stato utente . . . . .          | 19         |
| 2.3.1 Login e creazione della sessione . . . . .                         | 19         |
| 2.3.2 Recupero dell'utente corrente . . . . .                            | 20         |
| 2.3.3 Registrazione e inizializzazione del profilo . . . . .             | 21         |
| 2.4 Gestione delle viste e delle interazioni . . . . .                   | 22         |
| 2.4.1 Routing, navigazione globale e componenti riutilizzabili . . . . . | 22         |
| 2.4.2 Home . . . . .                                                     | 25         |
| 2.4.3 Annunci . . . . .                                                  | 29         |
| 2.4.4 Creazione di un nuovo annuncio . . . . .                           | 34         |
| 2.4.5 Creazione dell'annuncio nel backend . . . . .                      | 36         |
| 2.4.6 Modifica di un annuncio esistente . . . . .                        | 37         |
| 2.4.7 Gestione delle disponibilità nell'annuncio . . . . .               | 37         |
| 2.4.8 Partecipazione ad un annuncio e la logica di join . . . . .        | 38         |
| 2.4.9 Controlli preliminari . . . . .                                    | 38         |

|          |                                                                         |           |
|----------|-------------------------------------------------------------------------|-----------|
| 2.4.10   | Join dell'annuncio . . . . .                                            | 39        |
| 2.4.11   | Match: dal join alla validazione, con stato pending/confirmed . . . . . | 40        |
| 2.4.12   | Collegamento tra la vista Match e la dashboard amministrativa . . . . . | 44        |
| 2.4.13   | Chat tra utenti e continuità dopo il match . . . . .                    | 45        |
| 2.4.14   | Wallet: saldo, riscatti e storico movimenti . . . . .                   | 47        |
| 2.4.15   | Profilo utente, modifica dati e cronologia personale . . . . .          | 51        |
| 2.4.16   | Validazione e trasferimento crediti: completamento del ciclo . . . . .  | 52        |
| 2.4.17   | Gestione dello storico e tracciabilità delle interazioni . . . . .      | 54        |
| 2.5      | Accessibilità nell'implementazione . . . . .                            | 55        |
| 2.6      | Dashboard amministrativa come sottosistema di controllo . . . . .       | 56        |
| 2.6.1    | Autenticazione amministratore . . . . .                                 | 56        |
| 2.6.2    | Visualizzazione dei match . . . . .                                     | 57        |
| 2.6.3    | Validazione del match . . . . .                                         | 58        |
| 2.7      | Aspetti implementativi aggiuntivi . . . . .                             | 59        |
| 2.7.1    | Configurazione delle API di CORS e della sessione PHP . . . . .         | 59        |
| 2.7.2    | Recupero esteso e aggiornamento del profilo utente . . . . .            | 60        |
| 2.7.3    | Implementazione backend della chat . . . . .                            | 62        |
| 2.7.4    | Logout e chiusura esplicita della sessione . . . . .                    | 63        |
| <b>3</b> | <b>Validazione del progetto</b>                                         | <b>65</b> |
| 3.1      | Validazione funzionale . . . . .                                        | 65        |
| 3.2      | Validazione tecnica . . . . .                                           | 66        |
| 3.3      | Limiti del prototipo e validazione educativa . . . . .                  | 66        |
| 3.4      | Validazione sul campo: focus group nelle scuole . . . . .               | 67        |
| 3.4.1    | Analisi dei risultati del focus group . . . . .                         | 68        |
| 3.4.2    | Comprensione del modello e usabilità percepita . . . . .                | 70        |
| 3.4.3    | Valutazione del sistema di crediti . . . . .                            | 71        |
| 3.4.4    | Propensione alla partecipazione e dinamiche tra pari . . . . .          | 71        |
| 3.4.5    | Integrazione con lo spazio fisico di studio . . . . .                   | 71        |
| 3.4.6    | Sintesi dei risultati e implicazioni progettuali . . . . .              | 72        |
| 3.4.7    | Considerazioni sui risultati . . . . .                                  | 72        |
| 3.4.8    | Conclusioni sul focus group in ambito scolastico . . . . .              | 73        |
| 3.5      | Sicurezza e conformità al GDPR . . . . .                                | 73        |
|          | <b>Conclusioni</b>                                                      | <b>77</b> |

# Introduzione

Il sistema scolastico italiano si trova oggi ad affrontare sfide educative sempre più complesse, che non riguardano soltanto l'apprendimento disciplinare, ma toccano anche la sfera sociale, relazionale e culturale dei giovani. Fenomeni come la dispersione scolastica, ancora diffusa soprattutto nei contesti socio-economici più fragili e il numero elevato di giovani NEET (Not in Education, Employment or Training) [10], mostrano come l'accesso all'istruzione non sia sufficiente a garantire percorsi di crescita inclusivi. Per molti studenti, infatti, la scuola non riesce a rappresentare quello spazio di continuità educativa che dovrebbe sostenere il loro sviluppo, e le famiglie, spesso assorbite dai ritmi quotidiani, non sempre riescono a fornire un adeguato supporto pomeridiano allo studio.

Il rendimento scolastico risulta spesso correlato alle condizioni economiche delle famiglie, che determinano la possibilità o meno di accedere a risorse aggiuntive come ripetizioni private, doposcuola o spazi dedicati allo studio[8]. Allo stesso tempo, nel territorio mancano luoghi accessibili e accoglienti dove i ragazzi possano sentirsi accompagnati, motivati e responsabilizzati. Andando oltre alla semplice disponibilità di un ambiente fisico, l'esperienza educativa, infatti, richiede la presenza di adulti competenti, capaci di favorire processi di autonomia, di sostegno e di relazione significativa.

Negli ultimi anni, la riflessione pedagogica ha riconosciuto nella peer education e nel peer tutoring due strategie efficaci per promuovere partecipazione, responsabilizzazione e inclusione.[12] L'apprendimento tra pari valorizza il fatto che la conoscenza nasca e si consolidi in un contesto sociale in cui studenti della stessa fascia d'età, condividendo linguaggi, vissuti ed esperienze, possano aiutarsi reciprocamente nel comprendere contenuti, sviluppare nuove competenze e rafforzare il proprio senso di autoefficacia.

Il peer tutoring, in particolare, non si limita alla trasmissione di nozioni, ma implica capacità organizzative, comunicative e relazionali che permettono al tutor di consolidare ciò che sa e allo studente di sentirsi accolto in un contesto meno giudicante.

Numerosi studi mostrano come queste pratiche favoriscano non solo il miglioramento scolastico, ma anche la crescita dell'autostima, della motivazione e della partecipazione sociale dei ragazzi coinvolti.[5]

È proprio a partire da queste esigenze che nasce il Progetto GOAT, un'iniziativa che mira a creare uno spazio educativo ibrido, in parte fisico e in parte digitale, dove studio, relazione e partecipazione si intrecciano. L'idea è quella di costruire un ambiente accessibile e accogliente, in cui i ragazzi possano ricevere delle *"ripetizioni"* da altri ragazzi, magari più ferrati in una materia in cui la persona che richiede la lezione ha una lacuna, proporre le proprie competenze anche in altri ambiti apparte quello scolastico come quello artistico oppure quello musicale e inserirsi in una comunità in cui ciascuno può dare e ricevere aiuto. L'applicazione GOAT permette di organizzare questa rete di scambi e incontri, facilitando il matching tra studenti e tutor e incentivando la responsabilizzazione attraverso un sistema di crediti che possono essere reinvestiti in ore di ripetizione, attività formative o voucher presso realtà territoriali.[6]

Il progetto si propone quindi non solo di ridurre le disuguaglianze educative, ma anche di offrire ai giovani un luogo in cui stare, riconoscersi, sperimentare e costruire relazioni significative.

Alla luce di questa cornice, il presente elaborato analizza il Progetto GOAT come strumento innovativo di apprendimento cooperativo, approfondendone la struttura tecnologica e il funzionamento operativo. L'obiettivo è mostrare come un ambiente educativo integrato fondato sulla cooperazione, sulla reciprocità e sulla valorizzazione delle competenze possa contribuire a formare cittadini più consapevoli, solidali e attivamente partecipi della vita comunitaria.

Questo elaborato si dividerà in 3 capitoli:

- Capitolo 1: Analisi del contesto e dei requisiti
- Capitolo 2: Progettazione e implementazione del sistema
- Capitolo 3: Validazione del progetto

# Capitolo 1

## Progetto GOAT: Analisi del contesto e dei requisiti

### 1.1 Contesto applicativo

Negli ultimi anni si è assistito a una progressiva diffusione di piattaforme digitali progettate per facilitare l'incontro tra persone e competenze. In molti casi, tali strumenti non si esauriscono nell'interazione online, ma sono finalizzati alla costruzione di relazioni che proseguono in contesti fisici, come avviene, ad esempio, nelle applicazioni di socializzazione e organizzazione di attività in presenza [11] o nelle piattaforme di scambio di competenze e apprendimento collaborativo. Questo fenomeno riguarda numerosi ambiti applicativi, ma assume una particolare rilevanza nei contesti educativi, formativi e collaborativi, in cui la possibilità di coordinare in modo efficace le interazioni tra individui rappresenta un fattore determinante per il successo delle attività svolte.

In ambito educativo e di apprendimento informale, la collaborazione tra pari, il tutoraggio e lo scambio di conoscenze avvengono spesso attraverso strumenti digitali non strutturati, come applicazioni di messaggistica per quanto riguarda l'organizzazione per incontri in cui avviene uno scambio di denaro in cambio di lezioni che a volte non portano la persona a capire al meglio le nozioni impartite dalla controparte pagata o piattaforme social come ad esempio su Instagram<sup>1</sup> o Tiktok<sup>2</sup>. Sebbene tali strumenti offrano un accesso immediato e informale alle informazioni, essi risultano limitate nel momento in cui si rende necessario organizzare delle attività vere e proprie, definendo dei confini e conservandone anche la propria traccia nel tempo. In particolare, la mancanza di

---

<sup>1</sup>Applicazione social del gruppo META

<sup>2</sup>Applicazione social spopolata negli ultimi anni e conosciuta per rendere facile la propria usufruzione grazie ai contenuti di tempo breve

strutture fisiche, che non siano i vari luoghi pubblici o biblioteche in cui però si hanno problemi a comunicare, perchè nel primo caso essendo pubblico non viene garantito il silenzio e nel secondo invece non si può parlare ad un livello di voce tale da avere una conversazione "normale", rende difficile distinguere tra accordi presi, cosa è stato realmente fatto di quell'accordo e attività effettivamente concluse con tutte e due le parti soddisfatte del lavoro svolto, che sia per la parte che richiede l'aiuto ma anche per la parte che dà aiuto.

Il progetto *GOAT* si inserisce in questo scenario con l'obiettivo di fornire uno strumento digitale capace di supportare e strutturare l'incontro tra utenti che richiedono supporto e utenti che offrono competenze. La piattaforma è concepita come un ambiente unico in cui tali interazioni non solo possono avvenire, ma vengono anche governate attraverso regole chiare e condivise, che ne disciplinano l'intero ciclo di vita, poichè, dopo che i due o più utenti raggiungono un accordo per fare un'attività di scambio di informazioni, questa attività verrà svolta all'interno di uno spazio fisico fornito dal comune e adibito solamente alle attività accordate all'interno dell'applicativo, con la supervisione di un educatore che a fine attività convaliderà l'attività così da autorizzare il trasferimento di crediti da un wallet all'altro, cosicchè non si verifichino inconvenienti come nel caso in cui l'attività non venga realmente svolta o che la controparte pagata si approfitti dell'utente "debole".

Come ho accennato prima, a differenza delle semplici bacheche digitali o di canali di comunicazione informali, GOAT introduce una modellazione esplicita delle interazioni. Ogni attività è infatti all'interno di un processo che parte dalla pubblicazione di un annuncio, prosegue con la partecipazione controllata di un altro utente e si conclude solo a seguito di una conferma esplicita data del supervisore situato in quel luogo. In questo modo, il sistema non si limita a facilitare il contatto tra le parti, ma assume un ruolo attivo nella definizione delle regole attraverso cui tale contatto si traduce in un'attività riconosciuta all'interno della piattaforma.

## 1.2 Analisi del problema

L'idea alla base del progetto GOAT nasce dall'osservazione di una professoressa del comune di Cesena, Martina Giunchi<sup>3</sup>, che "denuncia" una criticità

---

<sup>3</sup>Laureata in Lettere e Italianistica, docente delle discipline letterarie nella Scuola Secondaria, assegnista di ricerca INVALSI, specializzata nella didattica dell'italiano per stranieri (DITALS II livello), attiva nelle realtà di volontariato per i giovani del territorio cesenate

ricorrente nei contesti di apprendimento informale e di collaborazione tra pari: la difficoltà nel coordinare le interazioni e nel riconoscere il valore delle attività svolte. In assenza di strumenti dedicati, chi offre supporto e chi lo riceve tende a fare affidamento su accordi informali, che risultano spesso difficili da monitorare e da verificare.

Questa modalità operativa comporta una serie di problematiche. In primo luogo, non esiste una distinzione chiara tra la semplice intenzione di collaborare e la collaborazione effettivamente realizzata. Un accordo preso tramite messaggio non garantisce che l'attività venga portata a termine, né consente di stabilire se e quando ciò sia avvenuto. In secondo luogo, la mancanza di una struttura persistente impedisce la costruzione di uno storico affidabile delle collaborazioni, che potrebbe invece rappresentare una risorsa importante sia per gli utenti sia per la piattaforma stessa.

Un ulteriore problema riguarda l'attribuzione di valore alle attività svolte. In assenza di meccanismi di conferma e validazione, risulta difficile riconoscere formalmente l'impegno degli utenti che offrono supporto, riducendo così le possibilità di incentivare una partecipazione continuativa e responsabile.

Ma, tra i motivi principali che hanno spinto questo progetto a prendere forma, è il fatto che molti ragazzi, influenzati dalla propria condizione familiare, spesso economicamente non stabile, non riescono a permettersi attività extrascolastiche che spesso hanno un costo elevato, per aiutarli a capire al meglio le nozioni impartite dai professori nel contesto scolastico e questo li porta "a gettare la spugna", non impegnandosi a scuola, intraprendendo strade che portano poi all'abbandono scolastico e all'aumento statistico della criminalità[1]

Il progetto GOAT affronta queste criticità introducendo un modello basato su annunci e match. Ogni annuncio rappresenta una richiesta o un'offerta di competenze, mentre il match costituisce l'elemento che formalizza l'incontro tra due utenti. Tale incontro non produce effetti immediati, ma rimane in uno stato intermedio fino a quando non viene confermato. Questa scelta progettuale consente di distinguere chiaramente tra l'avvio di una collaborazione e la sua effettiva conclusione, introducendo un livello di controllo che rafforza l'affidabilità complessiva del sistema.

### 1.3 Attori del sistema e logica dei ruoli

Un aspetto caratterizzante del progetto GOAT riguarda la gestione dei ruoli degli utenti. A differenza di molte piattaforme tradizionali, che assegnano ruoli statici e permanenti, il sistema adotta un approccio dinamico, in cui il

ruolo dell'utente dipende dal contesto dell'interazione.

Gli utenti possono pubblicare annunci di due tipi principali: richieste o offerte. Nel momento in cui un altro utente partecipa a un annuncio, il suo ruolo viene determinato automaticamente in funzione del tipo di annuncio stesso. In questo modo, un utente può assumere il ruolo di tutor in una determinata interazione e quello di studente in un'altra, senza che ciò richieda configurazioni aggiuntive o profili separati.

Questa scelta progettuale consente di mantenere il sistema flessibile e inclusivo, evitando una categorizzazione rigida degli utenti e favorendo una maggiore partecipazione, così da portare un utente a cercare di dare il proprio sostegno alla comunità dando il proprio contributo con discipline che lui conosce e ricevendo aiuto nelle discipline (scolastiche e non) dove ha maggiore difficoltà e su cui lui vuole essere aiutato. Il ruolo non definisce l'identità dell'utente all'interno della piattaforma, ma descrive la funzione che egli svolge in una specifica interazione.

Accanto agli utenti è presente una figura amministrativa, che può essere un educatore socio-culturale o un docente che vuole contribuire alla causa, che svolge una funzione di controllo e validazione. L'amministratore non interviene nelle fasi operative dell'interazione, ma ha il compito di confermare la conclusione delle attività, garantendo che solo le collaborazioni effettivamente svolte producano effetti sul sistema, cioè, grazie alla convalida da parte dell'amministratore si ha realmente lo "scambio di denaro" che non è denaro vero e proprio, ma GOAT Coins <sup>4</sup>. Questo ruolo contribuisce a rafforzare l'affidabilità del processo, introducendo un elemento di supervisione che separa l'intenzione di collaborare dalla collaborazione effettiva.

## 1.4 Primo mockup del sistema

Prima di passare alla modellazione formale del sistema, è stato realizzato un primo mockup grafico tramite FlutterFlow<sup>5</sup>, fornito dalla professoressa Martina Giunchi, già citata in precedenza come ideatrice del progetto, con l'obiettivo di dare una prima forma visiva all'idea di GOAT prima ancora di

---

<sup>4</sup>Valuta utilizzata all'interno dell'applicativo per incentivare un utente ad essere attivo all'interno della piattaforma

<sup>5</sup>Piattaforma online per la progettazione e prototipazione di applicazioni mobile e web, disponibile all'indirizzo <https://goat-mockup.flutterflow.app/>

iniziare la progettazione tecnica vera e propria.

Questo mockup non rappresenta l'applicazione finale, né nelle schermate né nel flusso di navigazione: è stato pensato soltanto come punto di partenza per definire uno stile grafico di massima e farsi un'idea generale di come potesse presentarsi un'app dedicata allo scambio di competenze tra studenti. Molte delle scelte fatte in questa fase iniziale, sia a livello di interfaccia sia a livello di funzionalità, sono state poi rivedute e in parte stravolte durante la progettazione e l'implementazione descritte nel Capitolo 2, una volta chiariti meglio i requisiti reali del sistema.

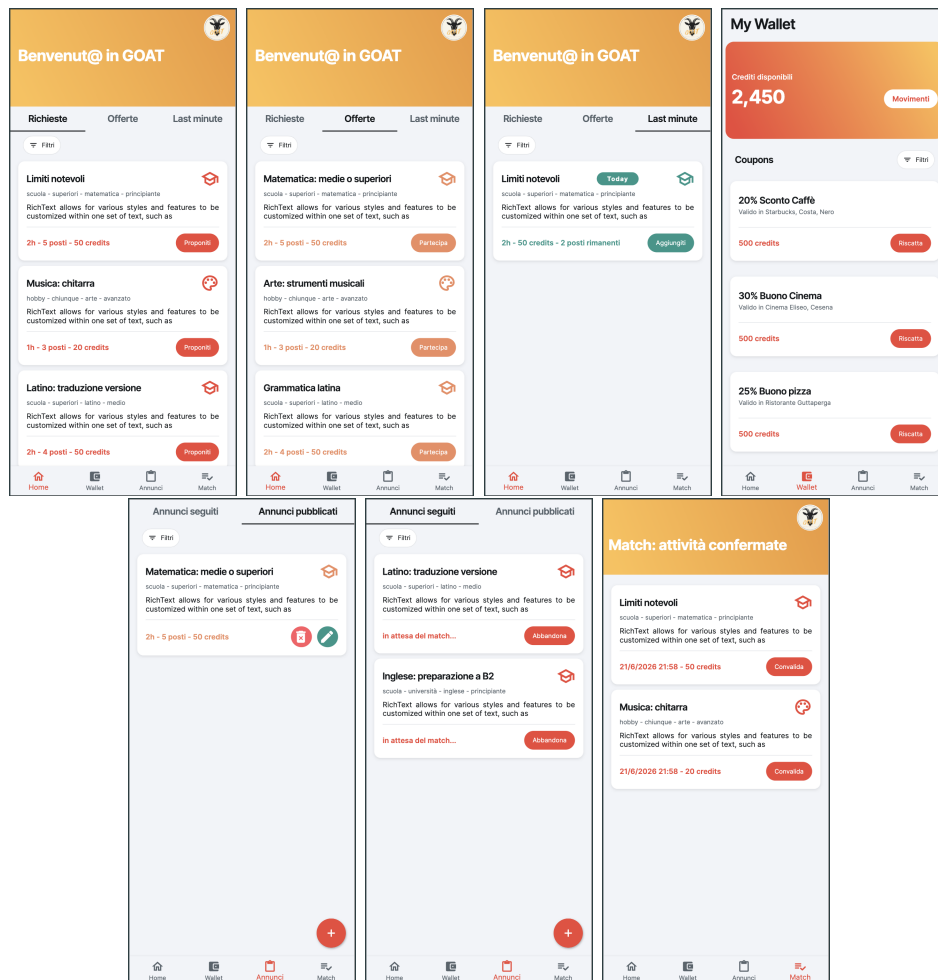


Figura 1.1: Screenshot del primo mockup del sistema GOAT realizzato con FlutterFlow

Nonostante le differenze rispetto al prodotto finale, questo primo mockup è stato comunque utile come punto di partenza concettuale, perché ha permesso di iniziare a ragionare visivamente sulle funzionalità principali che l'applicazione avrebbe dovuto offrire, prima di passare alla fase di analisi più strutturata.

## 1.5 Raccolta dei requisiti e analisi

Per descrivere in modo chiaro e formale il funzionamento della piattaforma, è stata adottata la modellazione UML (Unified Modeling Language)[4]. L'utilizzo dei diagrammi UML consente di rappresentare il sistema a un livello concettuale, facilitando la comprensione delle principali funzionalità offerte e delle relazioni tra gli attori e le entità coinvolte.

### 1.5.1 Diagramma dei casi d'uso

Il diagramma dei casi d'uso fornisce una visione d'insieme delle funzionalità del sistema dal punto di vista degli attori coinvolti. Esso mette in evidenza come l'utente possa interagire con la piattaforma pubblicando annunci, partecipando ad annunci esistenti e visualizzando lo stato delle collaborazioni, mentre l'amministratore interviene esclusivamente nella fase di validazione.

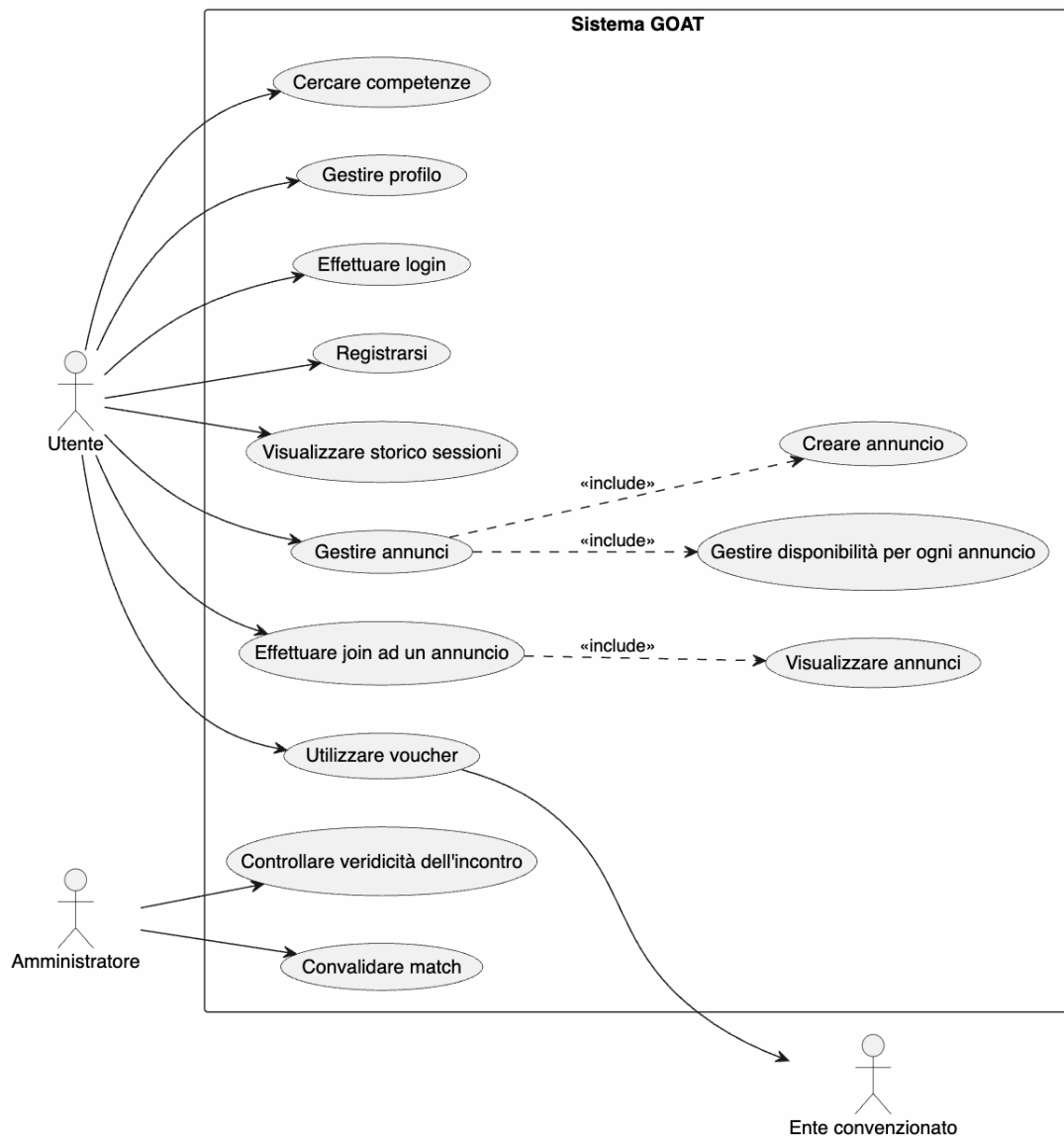


Figura 1.2: Diagramma dei casi d'uso del sistema GOAT

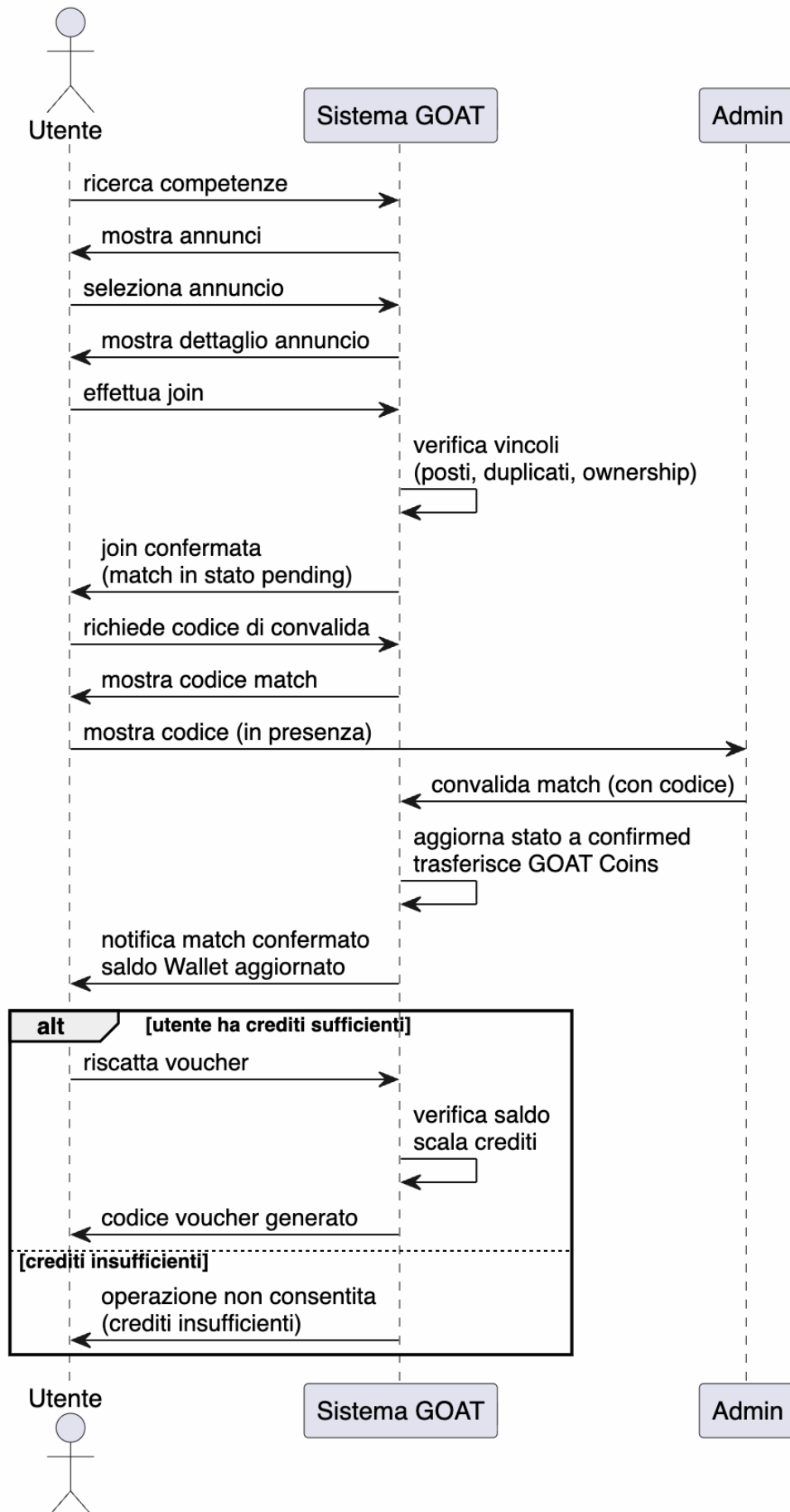
Il diagramma evidenzia una chiara separazione tra le funzionalità operative, affidate agli utenti, e le funzionalità di controllo, riservate all'amministratore.

### 1.5.2 Diagramma di sequenza

Il diagramma di sequenza rappresenta il flusso principale di interazione tra l'utente e il sistema GOAT nel caso di prenotazione di una sessione. Esiste anche il caso in cui l'utente non disponga dei GOAT Coins necessari per la

transazione.

La transazione viene, infine, convalidata dall'amministratore, garantendo un controllo minimo ma significativo sul processo.



## 1.6 Analisi del dominio

Prima di passare ai diagrammi e poi al codice, è utile fermarsi e definire con chiarezza i concetti principali su cui si basa GOAT. Questo serve a evitare ambiguità nelle fasi successive, perché termini che sembrano simili, come "partecipazione" e "match", in realtà indicano cose diverse all'interno del sistema.

Di seguito riporto un piccolo glossario delle entità principali del progetto, spiegando cosa rappresentano, quali informazioni portano con sé e come si collegano tra loro.

**Utente** È la persona che usa l'applicazione, sia quando chiede supporto sia quando lo offre. Non ha un ruolo fisso: dipende dalla situazione in cui si trova.

*Cosa porta con sé:* un identificativo, nome, email, password, il saldo di GOAT Coins e il ruolo che assume in quel momento.

*Come si collega:* un utente può creare più annunci, può partecipare ad annunci creati da altri, e può essere coinvolto in più match, sia come creatore che come partecipante.

**Annuncio** È la pubblicazione con cui un utente segnala una richiesta o un'offerta di competenze. È il punto da cui parte tutto il processo.

*Cosa porta con sé:* identificativo, tipo (richiesta o offerta), titolo, descrizione, posti disponibili, crediti associati e scadenza.

*Come si collega:* ogni annuncio ha un solo creatore, ma può generare più partecipazioni e più match, fino a quando i posti non finiscono.

**Partecipazione** È quando un utente decide di aderire a un annuncio pubblicato da qualcun altro. Non significa che la collaborazione sia conclusa, ma solo che è iniziata.

*Cosa porta con sé:* identificativo, ruolo assunto (tutor o studente) e stato.

*Come si collega:* collega un utente a un annuncio, e da questa azione nasce un match.

**Match** È il cuore del sistema: rappresenta il legame che si forma tra chi ha creato l'annuncio e chi vi ha partecipato. È importante perché introduce uno stato intermedio (*pending*) che separa il semplice accordo dalla collaborazione davvero conclusa e confermata (*confirmed*).

*Cosa porta con sé:* identificativo, codice di validazione, stato, data di creazione e data di conferma.

*Come si collega:* è legato a un annuncio, a chi lo ha creato e a chi vi

ha partecipato; quando viene confermato, fa scattare il trasferimento dei GOAT Coins.

**Voucher** È ciò che un utente può ottenere spendendo i propri GOAT Coins, da usare poi presso un ente convenzionato.

*Cosa porta con sé:* identificativo, codice, valore e se è già stato usato o no.

*Come si collega:* viene riscattato da un utente in cambio di crediti, ed è emesso da un ente convenzionato.

**Amministratore** È la figura di controllo, ad esempio un educatore, che non partecipa direttamente alle interazioni ma ha il compito di convalidare i match.

*Cosa porta con sé:* identificativo e credenziali di accesso.

*Come si collega:* convalida i match, facendoli passare da pending a confirmed.

**Ente convenzionato** È il soggetto esterno, che non fa parte di GOAT in senso stretto, presso cui un voucher può essere effettivamente usato.

*Cosa porta con sé:* identificativo e nome.

*Come si collega:* emette e verifica i voucher.

Questo glossario serve come punto di riferimento per tutto il resto della tesi: ogni scelta progettuale e tecnica descritta nel Capitolo 2 parte da queste entità, mantenendo lo stesso linguaggio usato qui.

## 1.7 Analisi dei requisiti

Dall'analisi del contesto applicativo e dalla modellazione concettuale del sistema emergono in modo chiaro i principali requisiti che la piattaforma GOAT deve soddisfare. Tali requisiti rappresentano il punto di accordo tra le esigenze individuate nella fase di analisi e le successive scelte progettuali e implementative.

Dal punto di vista funzionale, il sistema deve innanzitutto consentire agli utenti di pubblicare annunci in modo semplice e strutturato. La pubblicazione di un annuncio non rappresenta un'azione isolata, ma l'avvio di un processo che deve essere gestito e monitorato nel tempo. Per questo motivo, il requisito non si limita solo alla creazione di contenuti, ma contiene all'interno dell'annuncio informazioni rilevanti come il tipo di richiesta o offerta, il numero di partecipanti, l'eventuale scadenza temporale e il valore associato all'attività.

Un secondo requisito riguarda la partecipazione controllata agli annunci. Il sistema deve permettere agli utenti di aderire a un annuncio pubblicato da altri, ma tale adesione deve avvenire nel rispetto di vincoli ben definiti. In particolare, è necessario garantire la disponibilità dei posti e impedire comportamenti non corretti, come la partecipazione ai propri annunci. Questo requisito risponde all'esigenza di mantenere l'integrità del processo collaborativo e di prevenire incoerenze all'interno del sistema.

Un ulteriore requisito fondamentale è la possibilità di visualizzare lo stato delle collaborazioni in corso. Gli utenti devono poter distinguere chiaramente tra collaborazioni avviate ma non ancora concluse e collaborazioni effettivamente completate. Tale distinzione è essenziale per rendere il funzionamento della piattaforma comprensibile e trasparente, evitando ambiguità sullo stato delle attività ed è qui che si inserisce il requisito forse più caratterizzante dell'intero sistema: la presenza di un meccanismo di validazione dei match. Il sistema non deve considerare automaticamente conclusa una collaborazione nel momento in cui due utenti si accordano, ma deve introdurre uno stato intermedio che separi l'intenzione di collaborare dalla collaborazione effettivamente svolta. La validazione rappresenta quindi un passaggio obbligatorio che consente di confermare un match prima di renderlo definitivo e di attribuirgli effetti all'interno del sistema. Questo requisito risponde direttamente alle criticità individuate nella fase di analisi del problema, rafforzando l'affidabilità e la credibilità della piattaforma.

Accanto ai requisiti funzionali, emergono con altrettanta chiarezza i requisiti non funzionali del sistema[13]. In primo luogo, GOAT deve garantire una gestione coerente dei dati, evitando situazioni di inconsistenza tra le diverse entità coinvolte. La presenza di annunci, partecipazioni e match impone una struttura dei dati ben definita e regole precise di aggiornamento, affinché ogni operazione produca effetti coerenti sull'intero sistema.

Tra questi requisiti non funzionali abbiamo la tracciabilità delle operazioni. Ogni azione rilevante, come la pubblicazione di un annuncio, la partecipazione o la validazione di un match, deve poter essere ricostruita a posteriori. La tracciabilità non ha solo una funzione tecnica, ma contribuisce a rendere il sistema più affidabile.

Particolare attenzione deve inoltre essere posta all'usabilità dell'interfaccia. Il sistema è pensato per essere utilizzato anche da utenti non esperti dal punto di vista tecnologico; di conseguenza, l'interazione con la piattaforma deve risultare intuitiva, coerente e priva di complessità inutili. Questo requisito

influisce direttamente sulle scelte progettuali relative alla struttura delle viste e alla presentazione delle informazioni.

Un requisito trasversale riguarda la manutenibilità e l'evoluzione del sistema. GOAT è concepito come un progetto suscettibile di estensioni future, sia in termini di funzionalità, sia in termini di contesto applicativo. La struttura complessiva del sistema deve quindi favorire uno sviluppo incrementale, consentendo di introdurre nuove funzionalità senza compromettere quelle esistenti.

Nel loro insieme, i requisiti individuati delineano un sistema che non si limita a facilitare l'incontro tra utenti, ma che governa l'intero processo collaborativo, garantendone coerenza, controllo e tracciabilità. Nel capitolo successivo verrà mostrato come tali requisiti abbiano guidato le scelte progettuali e implementative, traducendosi in una soluzione software concreta.

Un ulteriore aspetto emerso dall'analisi dei requisiti riguarda il rapporto tra semplicità d'uso e controllo del processo. Da un lato, il sistema deve risultare immediato e comprensibile per l'utente finale, evitando configurazioni complesse o flussi poco intuitivi; dall'altro, deve essere sufficientemente strutturato da prevenire comportamenti impropri e garantire l'affidabilità delle interazioni. Questo equilibrio rappresenta uno dei principali vincoli progettuali del sistema GOAT.

In particolare, l'introduzione di stati intermedi, come lo stato **pending** (in attesa) per i match, risponde all'esigenza di rendere esplicito il ciclo di vita delle collaborazioni senza appesantire l'esperienza utente. Il requisito non è quindi quello di aumentare il numero di operazioni richieste all'utente, ma di rendere visibile e interpretabile ciò che avviene all'interno del sistema. La chiarezza dello stato delle attività diventa così un requisito funzionale e, allo stesso tempo, un requisito di usabilità.

Un ulteriore elemento di riflessione riguarda la separazione delle responsabilità all'interno del sistema. I requisiti individuati suggeriscono chiaramente la necessità di distinguere tra le funzionalità rivolte agli utenti finali e quelle dedicate al controllo e alla validazione. Questa distinzione non implica una complessità maggiore per l'utente, ma consente di isolare le operazioni più critiche in componenti specifiche del sistema, riducendo il rischio di errori e comportamenti incoerenti.

Infine, l'insieme dei requisiti mette in evidenza come GOAT non debba essere interpretata come una semplice applicazione di scambio. La piattaforma

deve essere in grado di accompagnare l'utente lungo tutte le fasi dell'interazione, fornendo feedback chiari e mantenendo una coerenza interna che renda il funzionamento del sistema comprensibile anche senza una conoscenza tecnica approfondita.

## Capitolo 2

# Progettazione e implementazione del sistema GOAT

### 2.1 Dal modello concettuale all'implementazione

Come anticipato nel Capitolo 1, il progetto GOAT nasce dall'esigenza di supportare in modo strutturato l'incontro tra domanda e offerta di competenze, garantendo tracciabilità, controllo e valorizzazione delle interazioni. A partire dai requisiti individuati, la progettazione del sistema si è concentrata sulla traduzione di tali esigenze in una soluzione software concreta.

L'implementazione ha seguito un approccio incrementale che è una metodologia in cui un sistema viene costruito e migliorato gradualmente, rilasciando piccole funzionalità che vengono integrate nel sistema esistente, permettendo adattamento continuo per ottenere un prodotto finale più robusto e allineato alle esigenze dell'utente.[2]

### 2.2 Architettura del sistema

Il sistema GOAT adotta un'architettura client-server articolata su tre livelli principali:

- frontend applicativo (Angular + Ionic)[3],
- backend applicativo (PHP),

- livello di persistenza (database relazionale MySQL).

Il frontend comunica con il backend esclusivamente tramite API REST, favorendo una separazione netta tra interfaccia utente e la logica applicativa.

### 2.2.1 Diagramma dei componenti

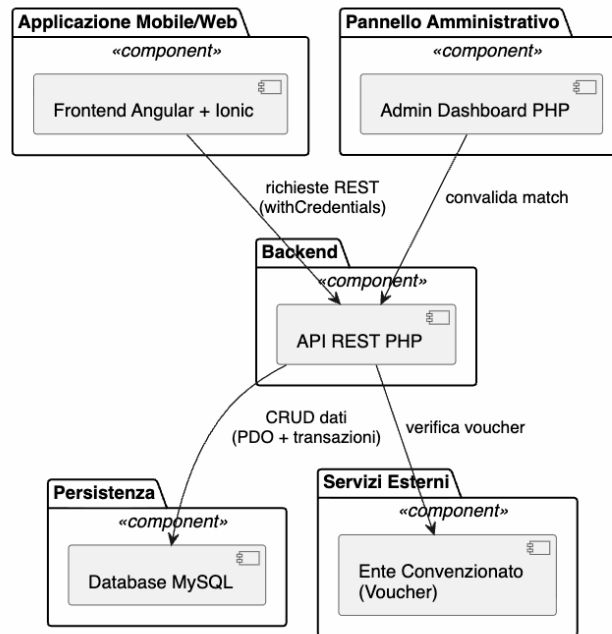


Figura 2.1: Diagramma dei componenti del sistema

Il diagramma dei componenti ha lo scopo di rappresentare l'architettura logica del sistema GOAT, evidenziando i principali sottosistemi software e le relazioni di dipendenza tra di essi. Attraverso questo diagramma è possibile ottenere una visione d'insieme del sistema, mettendo in evidenza la separazione delle responsabilità e le modalità di interazione tra le diverse componenti applicative.

In particolare, il sistema GOAT è strutturato secondo un'architettura a componenti [7], in cui il frontend (applicazione web e mobile) costituisce il punto di accesso per gli utenti e si occupa della gestione dell'interfaccia e dell'interazione con il sistema.

Il frontend comunica esclusivamente con il backend, che rappresenta il nucleo logico dell'applicazione e si occupa dell'elaborazione delle richieste, della gestione dei processi applicativi e del coordinamento delle operazioni principali.

Il backend interagisce con il database, responsabile della persistenza dei dati relativi agli utenti, agli annunci, alle sessioni e alle transazioni. Inoltre, il sistema prevede l'integrazione con servizi esterni, come l'ente convenzionato per la verifica e l'utilizzo dei voucher. Tali componenti esterne non fanno parte del sistema GOAT in senso stretto, ma risultano essenziali per il corretto funzionamento di alcune funzionalità chiave.

Infine, è prevista una componente dedicata all'amministrazione, che consente all'admin di interagire con il backend al fine di svolgere operazioni di controllo, in particolare la convalida delle transazioni. Questa scelta progettuale riflette l'intenzione di mantenere un ruolo amministrativo, focalizzato su funzioni di garanzia e supervisione, senza interferire con le dinamiche principali di interazione tra gli utenti.

### **2.2.2 Diagramma delle classi**

Il diagramma delle classi descrive la struttura del sistema, mostrando le principali entità informative e le relazioni che intercorrono tra di esse. Le classi fondamentali del sistema sono l'utente, l'annuncio, la partecipazione e la transazione (match).

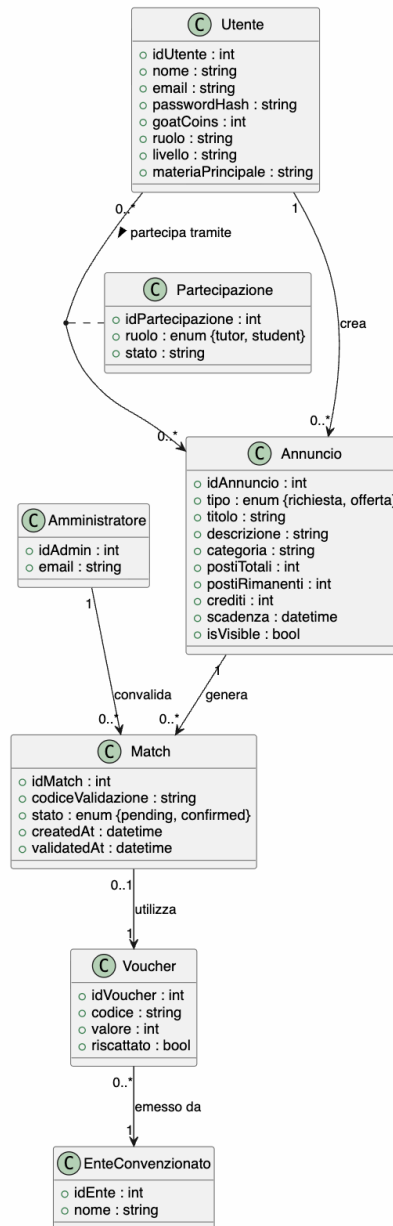


Figura 2.2: Diagramma delle classi del sistema

La distinzione tra partecipazione e match consente di separare concettualmente l'adesione a un annuncio dalla sua effettiva conclusione.

## 2.3 Gestione dell'autenticazione e dello stato utente

Un elemento fondamentale del sistema GOAT è la gestione dell'utente loggato. Tutte le funzionalità principali dell'applicazione dalla visualizzazione degli annunci alla partecipazione, fino alla validazione dei match e al Wallet presuppongono l'esistenza di una sessione utente attiva.

Dal punto di vista architetturale, si è scelto di adottare un meccanismo di autenticazione basato su sessione PHP, anziché su token JWT. Questa scelta è motivata da due fattori principali: da un lato la semplicità di integrazione con il backend PHP, dall'altro la presenza di una dashboard amministrativa sviluppata anch'essa in PHP, che può così condividere lo stesso meccanismo di autenticazione.

### 2.3.1 Login e creazione della sessione

Il processo di login avviene tramite una chiamata POST al backend, che verifica le credenziali e inizializza la sessione.

```
session_start();
require __DIR__ . '/../config/database.php';

$data = json_decode(file_get_contents('php://input'), true);
$email = $data['email'] ?? null;
$password = $data['password'] ?? null;

$stmt = $pdo->prepare("
    SELECT id, name, email, password_hash, goat_coins
    FROM users
    WHERE email = ?
    LIMIT 1
");
$stmt->execute([$email]);
$user = $stmt->fetch(PDO::FETCH_ASSOC);

if (!$user || !password_verify($password, $user['password_hash'])) {
    http_response_code(401);
    echo json_encode(['error' => 'Invalid credentials']);
    exit;
}
```

```
$_SESSION['user'] = [
    'id' => $user['id'],
    'name' => $user['name'],
    'email' => $user['email'],
];

echo json_encode(['success' => true]);
```

Listato 2.1: login.php: autenticazione e inizializzazione sessione

Una volta inizializzata la sessione, il cookie PHP viene automaticamente inviato al client e mantenuto nelle successive richieste, grazie all'opzione `withCredentials` utilizzata nel frontend.

### 2.3.2 Recupero dell'utente corrente

Per mantenere lo stato dell'utente sincronizzato tra le diverse viste, il frontend utilizza un servizio di autenticazione che contiene un `observable` contenente l'utente corrente.

```
me() {
    return this.http
        .get<User>(this.baseUrl + 'me.php', { withCredentials: true })
        .pipe(
            tap((user) => {
                this.userSubject.next(user);
                this.currentUser = user;
            })
        );
}
```

Listato 2.2: AuthService: recupero utente corrente

Il backend restituisce i dati dell'utente associato alla sessione attiva:

```
session_start();

if (!isset($_SESSION['user'])) {
    http_response_code(401);
    echo json_encode(['error' => 'Not authenticated']);
    exit;
}
```

```
echo json_encode($_SESSION['user']);
```

Listato 2.3: me.php: restituzione dati utente dalla sessione

Questo meccanismo consente a tutte le viste di reagire automaticamente a cambiamenti dello stato utente, come il login, il logout o l'aggiornamento del saldo dei GOAT Coins (crediti dell'app).

### 2.3.3 Registrazione e inizializzazione del profilo

Accanto al login, l'applicazione implementa anche un flusso di registrazione che consente la creazione di un nuovo utente direttamente dal frontend. La fase di signup non si limita a salvare delle credenziali, ma rappresenta il punto di ingresso nel sistema e quindi il momento in cui viene inizializzato il profilo di base dell'utente.

Nel frontend, la pagina di registrazione raccoglie nome, email e password e applica un controllo preliminare sui campi, richiedendo che tutti siano compilati e che la password abbia una lunghezza minima. Dopo la creazione dell'account, il sistema effettua automaticamente il login dell'utente appena registrato, evitando un passaggio ulteriore e rendendo più fluido l'accesso iniziale all'applicazione.

```
async doSignup() {
  this.error = '';

  const name = (this.name ?? '').trim();
  const email = (this.email ?? '').trim().toLowerCase();
  const pass = this.pass ?? '';

  if (!name || !email || pass.length < 8) {
    this.error = 'Compila tutti i campi (password 8).';
    return;
  }

  this.loading = true;
  try {
    await this.auth.signup(name, email, pass);
    await this.auth.login(email, pass);
    await this.auth.me();
    await this.router.navigateByUrl('/home', { replaceUrl: true });
  } finally {
    this.loading = false;
  }
}
```

```
}

```

Listato 2.4: SignupPage: registrazione con login automatico (frontend)

Dal punto di vista architetturale, questa scelta mantiene coerente la gestione dello stato: anche nel caso della registrazione, il **BehaviorSubject** contenente l'utente corrente viene aggiornato e tutte le viste dipendenti possono leggere immediatamente i dati del nuovo utente senza richiedere logiche separate.

## 2.4 Gestione delle viste e delle interazioni

La progettazione delle viste principali di GOAT non è stata affrontata come una semplice “impaginazione” dell'interfaccia.

In particolare, la scelta di Angular + Ionic ha permesso di implementare pagine standalone orientate a componenti riutilizzabili, mentre la presenza di un backend PHP con sessione ha imposto un'attenzione specifica alla gestione dell'autenticazione e alla corretta propagazione dello stato dell'utente tra viste diverse.

Nell'applicazione, le viste Home, Annunci, Wallet, Chat e Match non sono entità isolate, ma costituiscono un flusso unico che accompagna l'utente nella scoperta degli annunci, nella partecipazione o gestione delle proprie pubblicazioni, nella generazione del match e nella sua validazione, e infine nella valorizzazione dell'attività svolta tramite crediti e voucher nel Wallet. La continuità dell'esperienza utente è garantita da due aspetti fondamentali: da un lato una UI coerente (header e card uniformate, tab interne), dall'altro un backend che impone delle regole ben precise così da non creare ambiguità, rendendo la UX più accessibile (annunci scaduti non visibili, join non duplicabili, trasferimento crediti solo dopo conferma, ecc.).

### 2.4.1 Routing, navigazione globale e componenti riutilizzabili

Un aspetto importante dell'implementazione frontend riguarda la gestione centralizzata della navigazione. GOAT adotta il routing di Angular con pagine standalone caricate dinamicamente, in modo da mantenere le viste modulari e indipendenti tra loro.

Nel file di routing, ogni pagina principale viene associata a un percorso dedicato. Questa organizzazione consente sia una chiara separazione delle

responsabilità sia una più semplice evoluzione del progetto, perché nuove viste possono essere aggiunte senza alterare la struttura di quelle esistenti.

```
export const routes: Routes = [
  { path: '', redirectTo: 'login', pathMatch: 'full' },
  {
    path: 'login',
    loadComponent: () =>
      import('./pages/auth/login/login.page').then((m) =>
        m.LoginPage),
  },
  {
    path: 'home',
    canActivate: [
      () => import('./core/guards/auth.guard').then((g) =>
        g.canActivateAuth),
    ],
    loadComponent: () =>
      import('./pages/home/home.page').then((m) => m.HomePage),
  },
  {
    path: 'signup',
    loadComponent: () =>
      import('./pages/auth/signup/signup.page').then((m) =>
        m.SignupPage),
  },
  {
    path: 'profile',
    loadComponent: () =>
      import('./pages/profile/profile.page').then((m) =>
        m.ProfilePage),
  },
  {
    path: 'profile-edit',
    loadComponent: () =>
      import('./pages/profile-edit/profile-edit.page').then(
        (m) => m.ProfileEditPage
      ),
  },
  {
    path: 'new-announcement',
    loadComponent: () =>
      import('./pages/new-announcement/new-announcement.page').then(
        (m) => m.NewAnnouncementPage
      )
  }
]
```

```

    ),
  },
  {
    path: 'wallet',
    loadComponent: () =>
      import('./pages/wallet/wallet.page').then((m) => m.WalletPage),
  },
  {
    path: 'annunci',
    loadComponent: () =>
      import('./pages/annunci/annunci.page').then((m) =>
        m.AnnunciPage),
  },
  {
    path: 'match',
    loadComponent: () =>
      import('./pages/match/match.page').then((m) => m.MatchPage),
  },
  {
    path: 'chat',
    canActivate: [
      () => import('./core/guards/auth.guard').then((g) =>
        g.canActivateAuth),
    ],
    loadComponent: () =>
      import('./pages/chat/chat.page').then((m) => m.ChatPage),
  },
  {
    path: 'chat/:id',
    canActivate: [
      () => import('./core/guards/auth.guard').then((g) =>
        g.canActivateAuth),
    ],
    loadComponent: () =>
      import('./pages/chat-room/chat-room.page').then((m) =>
        m.ChatRoomPage),
  },
  {
    path: 'edit-announcement/:id',
    loadComponent: () =>
      import('./pages/edit-announcement/edit-announcement.page').then(
        (m) => m.EditAnnouncementPage
      ),
  },
},

```

```
];
```

Listato 2.5: app.routes.ts: definizione delle rotte principali (frontend)

La navigazione persistente tra le aree principali è resa tramite una bottom bar comune. Questa scelta evita di duplicare elementi di interfaccia nelle singole pagine e rafforza la percezione di continuità dell'app. Inoltre, la barra viene nascosta nelle schermate di autenticazione e nella singola conversazione di chat, così da ridurre distrazioni e lasciare più spazio utile al contenuto.

```
<ion-tab-bar>
  <ion-tab-button routerLink="/home">
    <ion-label>Home</ion-label>
  </ion-tab-button>
  <ion-tab-button routerLink="/wallet">
    <ion-label>Wallet</ion-label>
  </ion-tab-button>
  <ion-tab-button routerLink="/annunci">
    <ion-label>Annunci</ion-label>
  </ion-tab-button>
  <ion-tab-button routerLink="/match">
    <ion-label>Match</ion-label>
  </ion-tab-button>
  <ion-tab-button routerLink="/chat">
    <ion-label>Chat</ion-label>
  </ion-tab-button>
</ion-tab-bar>
```

Listato 2.6: app.component.html: navigazione globale tra le viste principali

Un ulteriore elemento riutilizzabile presente in più viste è il collegamento rapido al profilo utente, implementato come componente dedicato. In questo modo, Home, Match e Wallet possono mostrare in alto lo stesso accesso al profilo, mantenendo uniformità visiva e riducendo duplicazioni nel codice.

### 2.4.2 Home

La Home è stata progettata come pagina di esplorazione a scorrimento con dei filtri rappresentati dalle tab. La scelta delle tre tab (Richieste, Offerte, Last minute) deriva da un vincolo di usabilità: l'utente deve poter distinguere rapidamente tra annunci che richiedono supporto, annunci che offrono un servizio, e contenuti urgenti che scadono in giornata. Questo riduce sia il rischio di creare ambiguità sulle intenzioni che ha quell'annuncio sia la probabilità di

interazioni non intenzionali.

Sul piano tecnico, la Home mantiene uno stato interno `activeTab` che determina il parametro `type` passato al backend. La logica di caricamento è che ad ogni cambio tab richiama un metodo di fetch che aggiorna `announcements` e disattiva il loader.

```
activeTab: 'richiesta' | 'offerta' | 'lastminute' = 'richiesta';
announcements: Announcement[] = [];
loading = true;

changeTab(tab: 'richiesta' | 'offerta' | 'lastminute') {
  this.activeTab = tab;
  this.loadAnnouncements();
}

loadAnnouncements() {
  this.loading = true;
  this.announcementsService
    .getAnnouncements(this.activeTab)
    .subscribe((res) => {
      this.announcements = res.data;
      this.loading = false;
    });
}
```

Listato 2.7: Gestione tab e caricamento annunci in Home (frontend)

Il caricamento avviene tramite un servizio Angular che utilizza la logica HTTP e delle query params. La presenza di un service dedicato evita duplicazione di codice e rende più facile modificare in futuro il richiamo delle API.

```
getAnnouncements(
  type: 'richiesta' | 'offerta' | 'lastminute'
): Observable<{ success: boolean; data: Announcement[] }> {
  const params = new HttpParams().set('type', type);
  return this.http.get<{ success: boolean; data: Announcement[] }>(
    this.baseUrl + 'announcements.php',
    { params, withCredentials: true }
  );
}
```

Listato 2.8: AnnouncementsService: recupero annunci per tab (frontend)

È importante notare l'opzione `withCredentials`. In GOAT l'autenticazione si basa sulla sessione PHP, senza `withCredentials` il browser non invierebbe il cookie di sessione, portando il backend a vedere l'utente come non autenticato anche dopo il login.

Sul lato backend, `announcements.php` implementa il filtro già accennato prima. Il criterio di base è che deve mostrare solo annunci visibili e non scaduti cioè che il parametro `is_visible` deve essere settato a 1 per essere visibile. La tab "lastminute" non è un tipo nel database (il campo `type` è richiesta/offerta), ma seleziona annunci con `expires_at` in data odierna, ordinandoli in modo che quelli più prossimi alla scadenza appaiano prima, guardando l'orario.

```
$type = $_GET['type'] ?? null;

$allowed = ['richiesta', 'offerta', 'lastminute'];
if (!in_array($type, $allowed, true)) {
    http_response_code(400);
    echo json_encode(['error' => 'Invalid type']);
    exit;
}

$sql = "
    SELECT
        id, type, title, subtitle, description, category,
        duration_hours, total_spots, remaining_spots, credits,
        event_date, event_time, expires_at,
        cta_label, cta_action, priority, created_at
    FROM announcements
    WHERE is_visible = 1
        AND (expires_at IS NULL OR expires_at >= NOW())
";

$params = [];

if ($type === 'lastminute') {
    $sql .= " AND expires_at IS NOT NULL AND DATE(expires_at) =
        CURDATE() ";
    $sql .= " ORDER BY expires_at ASC, priority DESC, created_at DESC
        ";
} else {
    $sql .= " AND type = :type ";
    $params[':type'] = $type;
```

```
$sql .= " ORDER BY priority DESC, created_at DESC ";
}

$stmt = $pdo->prepare($sql);
$stmt->execute($params);
$rows = $stmt->fetchAll(PDO::FETCH_ASSOC);

echo json_encode(['success' => true, 'data' => $rows]);
```

Listato 2.9: announcements.php: filtro per tab e gestione scadenze (backend)

Questo endpoint costituisce il cuore della Home, tutte le decisioni di visibilità e urgenza sono imposte dal backend, così da non dipendere da logiche facilmente aggirabili o incoerenti tra dispositivi diversi.

Un ulteriore requisito rilevante del progetto emerso durante l'implementazione è evitare che l'utente veda in Home gli annunci creati da sé stesso. Poiché GOAT modella la creazione di un annuncio come un'azione che inizia un processo, mostrare i propri annunci nella Home di esplorazione avrebbe generato confusione e potenziali tentativi di join su annunci propri. A livello backend, questa esigenza si risolve aggiungendo una condizione basata sul `created_by` letto dalla sessione:

```
if (isset($_SESSION['user'])) {
    $uid = (int) $_SESSION['user']['id'];
    $sql .= " AND created_by <> :uid ";
    $params[':uid'] = $uid;
}
```

Listato 2.10: Esclusione annunci creati dall'utente corrente (backend)

In questo modo, la Home rimane coerente cioè è una bacheca per trovare annunci di altri utenti.

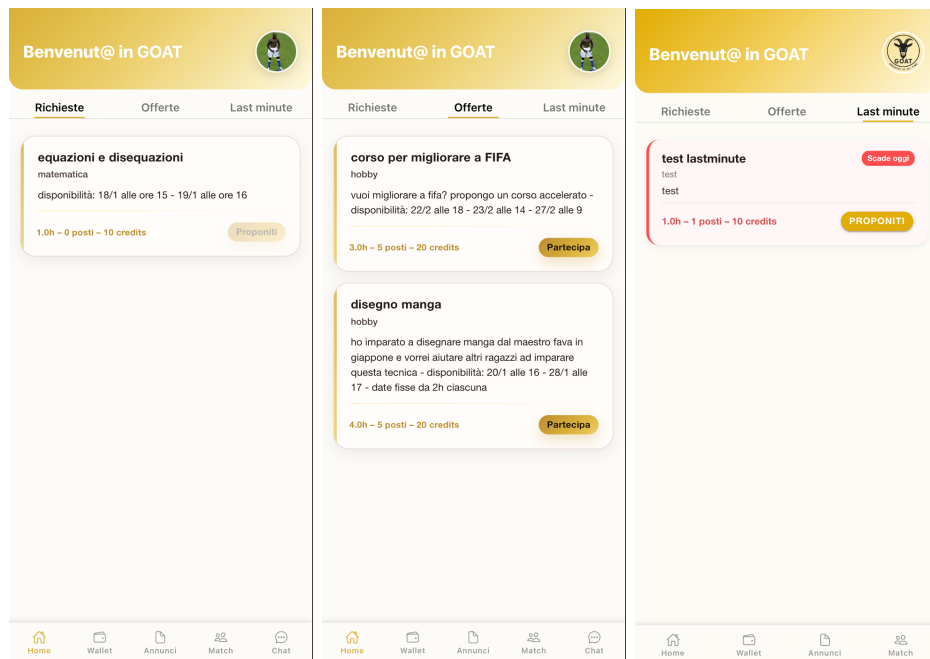


Figura 2.3: Screen delle varie tab all'interno della home

### 2.4.3 Annunci

La vista Annunci è stata progettata come un “area personale” dell’utente, distinta dalla Home. La Home è orientata alla scoperta, mentre Annunci è orientata alla gestione, qui l’utente vede gli annunci che ha pubblicato e quelli a cui ha aderito, con azioni coerenti con il proprio ruolo nel processo.

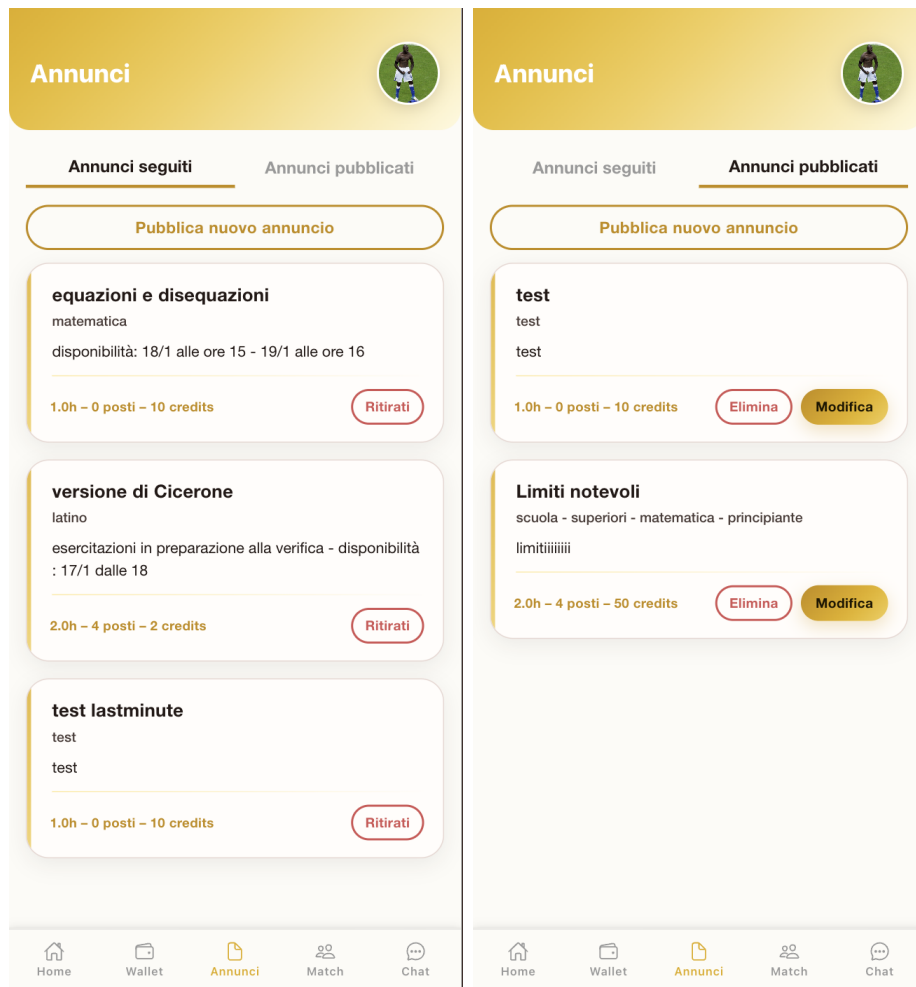


Figura 2.4: Screen della vista annunci

L'interfaccia presenta una tab interna a due stati: `pubblicati` e `seguiti`. La tab selezionata determina quale endpoint chiamare. Il comportamento è simile alla Home, ma con una differenza cruciale: non si filtra per tipologia (richiesta/offerta), bensì per relazione utente-annuncio.

```
activeTab: 'seguiti' | 'pubblicati' = 'pubblicati';
annunci: any[] = [];
loading = false;

ngOnInit() {
  this.load();
}

load() {
```

```
this.loading = true;

const req =
  this.activeTab === 'pubblicati'
    ? this.service.getPublished()
    : this.service.getFollowed();

req.subscribe({
  next: (res) => {
    this.annunci = res.data;
    this.loading = false;
  },
  error: () => {
    alert('Errore caricamento annunci');
    this.loading = false;
  },
});
}

switchTab(tab: 'seguiti' | 'pubblicati') {
  this.activeTab = tab;
  this.load();
}
```

Listato 2.11: AnnunciPage: caricamento differenziato per tab (frontend)

La pagina, inoltre, abilita azioni diverse in base alla tab. Per gli annunci pubblicati l'utente può modificare o eliminare; per gli annunci seguiti può ritirarsi. Questa differenza viene resa esplicita anche in HTML con direttive condizionali.

```
<div class="actions">
  <ion-button
    *ngIf="activeTab === 'pubblicati'"
    color="danger"
    (click)="delete(annuncio.id)">
    Elimina
  </ion-button>

  <ion-button
    *ngIf="activeTab === 'pubblicati'"
    color="warning"
    (click)="edit(annuncio.id)">
    Modifica
```

```

</ion-button>

<ion-button
  *ngIf="activeTab === 'seguiti'"
  color="danger"
  (click)="leave(annuncio.id)">
  Ritirati
</ion-button>
</div>

```

Listato 2.12: Azioni condizionate in Annunci: published vs followed (frontend)

Dal punto di vista applicativo, “Elimina” non rimuove fisicamente la riga dal database, ma applica una cancellazione logica impostando `is_visible = 0`. Questa scelta è stata adottata per evitare perdita di informazioni storiche e per mantenere coerenza con la logica di scadenza e visibilità (un annuncio può diventare non visibile per varie ragioni).

```

$upd = $pdo->prepare("
  UPDATE announcements
  SET is_visible = 0
  WHERE id = ?
");
$upd->execute([$announcementId]);

echo json_encode(['success' => true]);

```

Listato 2.13: delete\_announcement.php: cancellazione logica (backend)

L’endpoint verifica anche l’ownership: un utente può eliminare solo annunci creati da lui. Nel progetto, il controllo di ownership rimane un vincolo essenziale per evitare che un utente manipoli annunci non propri.

Per gli annunci seguiti, invece, l’azione “Ritirati” modifica la tabella di partecipazione e ripristina il numero di posti disponibili. Questo comportamento richiama la logica inversa della join: se l’utente esce, il sistema deve incrementare i `remaining_spots`.

```

$pdo->beginTransaction();
try {
  $del = $pdo->prepare("
    DELETE FROM announcement_participants
    WHERE announcement_id = ? AND user_id = ?
  ");
  $del->execute([$announcementId, $userId]);
}

```

```
$upd = $pdo->prepare("
    UPDATE announcements
    SET remaining_spots = remaining_spots + 1
    WHERE id = ?
");
$pdo->execute([$announcementId]);

$pdo->commit();
echo json_encode(['success' => true]);
} catch (Throwable $e) {
    $pdo->rollBack();
    http_response_code(500);
    echo json_encode(['error' => 'Leave failed']);
}
```

Listato 2.14: leave\_announcement.php: ritiro e ripristino posti (backend)

Il fatto che la vista Annunci chiami gli endpoint `published_announcements.php` e `followed_announcements.php` consente di mantenere la Home pulita; La Home mostra le opportunità, Annunci, invece mostra le responsabilità e le scelte già compiute. Questa distinzione riduce errori utente e mantiene la UI coerente con la semantica del processo.

### 2.4.4 Creazione di un nuovo annuncio

Figura 2.5: Screen della vista nuovo annuncio

La vista *Nuovo annuncio* consente all'utente di avviare un nuovo processo pubblicando un annuncio nuovo. A differenza di un semplice form di inserimento dati, questa vista incorpora una serie di scelte progettuali volte a garantire coerenza informativa, prevenzione degli errori e allineamento con la logica backend.

A livello frontend, la pagina è implementata come componente Angular standalone e utilizza `ngModel` per il two-way binding dei dati

```
form = {  
  type: 'richiesta' as 'richiesta' | 'offerta',  
  title: '',  
  subtitle: '',  
  description: '',  
  category: '',  
  duration_hours: 1,  
  total_spots: 1,  
  credits: 10,  
  event_date: null as string | null,  
  event_time: null as string | null,  
  is_lastminute: false,  
};
```

Listato 2.15: NewAnnouncementPage: struttura del form (frontend)

Un aspetto rilevante riguarda la gestione delle date. I componenti `IonDatetime` restituiscono valori in formato ISO 8601 completo<sup>1</sup>, che non sono direttamente compatibili con i tipi SQL DATE e TIME. Per questo motivo, prima dell'invio al backend, il payload viene normalizzato.

```
const payload = {  
  ...this.form,  
  event_date: this.form.event_date  
    ? this.form.event_date.split('T')[0]  
    : null,  
  event_time: this.form.event_time  
    ? this.form.event_time.split('T')[1]?.substring(0, 8)  
    : null,  
};
```

Listato 2.16: Normalizzazione data e ora evento prima dell'invio (frontend)

Questa scelta evita errori di formato lato backend e definisce una chiara separazione di responsabilità: il frontend si occupa della presentazione e della normalizzazione, mentre il backend si concentra sulla validazione e persistenza.

Dal punto di vista dell'interazione, l'invio del form disabilita temporaneamente il pulsante di submit e mostra un feedback visivo, prevenendo invii multipli accidentali.

---

<sup>1</sup>Esempio: data e ora 2026-01-03T17:11Z

```
<ion-button expand="block" (click)="submit()" [disabled]="loading">
  {{ loading ? 'Salvataggio' : 'Pubblica annuncio' }}
</ion-button>
```

Listato 2.17: Pulsante di submit con gestione stato (frontend)

### 2.4.5 Creazione dell'annuncio nel backend

La creazione dell'annuncio è gestita dal file `new_announcements.php`.

Un primo aspetto rilevante riguarda l'etichetta della CTA (`cta_label`). Questa non viene inserita manualmente dall'utente, ma viene determinata automaticamente in base al tipo di annuncio, evitando incoerenze nell'interfaccia

```
$CTA_BY_TYPE = [
    'richiesta' => 'Proponiti',
    'offerta' => 'Partecipa',
];

$type = $data['type'] ?? null;
$ctaLabel = $CTA_BY_TYPE[$type] ?? null;
```

Listato 2.18: CTA automatica in base al tipo annuncio (backend)

L'inserimento vero e proprio avviene tramite prepared statement, includendo l'utente creatore e inizializzando i posti disponibili:

```
INSERT INTO announcements (
    type, title, subtitle, description, category,
    duration_hours, total_spots, remaining_spots,
    credits, event_date, event_time, expires_at,
    cta_label, created_by
) VALUES (
    :type, :title, :subtitle, :description, :category,
    :duration_hours, :total_spots, :total_spots,
    :credits, :event_date, :event_time, :expires_at,
    :cta_label, :created_by
)
```

Listato 2.19: Inserimento annuncio nel database (backend)

Questa fase conclude la creazione dell'annuncio e rende immediatamente disponibile il contenuto nella Home degli altri utenti.

### 2.4.6 Modifica di un annuncio esistente

La vista di modifica dell'annuncio rappresenta un'estensione naturale della creazione, ma introduce vincoli aggiuntivi. Un annuncio può essere modificato solo dal suo creatore e solo se non è stato ancora validato tramite match.

Dal punto di vista frontend, la pagina *Edit Annuncio* recupera inizialmente i dati dal backend e popola il form:

```
ngOnInit() {  
  const id = this.route.snapshot.paramMap.get('id');  
  this.service.getAnnouncement(id!).subscribe(res => {  
    this.form = res.data;  
  });  
}
```

Listato 2.20: Caricamento annuncio in modifica (frontend)

Il backend, nel file `get_announcement.php`, verifica che l'annuncio esista e che l'utente corrente ne sia il proprietario:

```
SELECT id FROM announcements  
WHERE id = ? AND created_by = ?  
LIMIT 1
```

Listato 2.21: Verifica ownership annuncio (backend)

Questa verifica è fondamentale per evitare modifiche non autorizzate e garantire la coerenza dei dati.

### 2.4.7 Gestione delle disponibilità nell'annuncio

Nel progetto è presente anche una gestione delle disponibilità orarie associate all'annuncio. Oltre ai campi principali dell'evento, il frontend consente infatti di aggiungere una lista di slot composti da data, ora di inizio e ora di fine. Questa scelta è utile perché non tutti gli annunci descrivono un singolo appuntamento; in alcuni casi è più corretto rappresentare un insieme di fasce disponibili tra cui organizzare l'incontro.

Dal punto di vista implementativo, il frontend mantiene un array dedicato di disponibilità e, al momento dell'invio, normalizza anche questi dati prima di inserirli nel payload destinato al backend.

```
availabilities: AnnouncementAvailability[] = [];  
  
addAvailabilitySlot() {
```

```
const today = new Date().toISOString().split('T')[0];
this.availabilities.push({
  date: today,
  start_time: '09:00',
  end_time: '10:00',
});
}

const payload = {
  ...this.form,
  availabilities: this.availabilities.map((a) => ({
    date: this.normalizeDate(a.date),
    start_time: this.normalizeTime(a.start_time),
    end_time: this.normalizeTime(a.end_time),
  })),
};
```

Listato 2.22: NewAnnouncementPage: gestione delle disponibilità multiple (frontend)

Questa parte arricchisce il modello dell'annuncio, perché consente di distinguere tra la descrizione generale della richiesta/offerta e le reali finestre temporali in cui l'utente è disponibile.

### 2.4.8 Partecipazione ad un annuncio e la logica di join

La partecipazione ad un annuncio rappresenta uno dei momenti più delicati del sistema, poiché implica una modifica simultanea di più entità: annuncio, partecipazione e match.

### 2.4.9 Controlli preliminari

Prima di consentire la join, il backend verifica:

- che l'annuncio esista e sia visibile;
- che non sia scaduto;
- che vi siano posti disponibili;
- che l'utente non sia il creatore;
- che l'utente non abbia già partecipato.

```
if ($announcement['created_by'] === $userId) {
    http_response_code(403);
    echo json_encode(['error' => 'Cannot join own announcement']);
    exit;
}

$check = $pdo->prepare("
    SELECT 1 FROM announcement_participants
    WHERE announcement_id = ? AND user_id = ?
");
$check->execute([$announcementId, $userId]);
if ($check->fetch()) {
    http_response_code(409);
    echo json_encode(['error' => 'Already joined']);
    exit;
}
```

Listato 2.23: join\_announcement.php: controlli preliminari

### 2.4.10 Join dell'annuncio

```
$pdo->beginTransaction();

try {
    $pdo->prepare("
        INSERT INTO announcement_participants
        (announcement_id, user_id, role, status)
        VALUES (?, ?, ?, 'pending')
    ")->execute([$announcementId, $userId, $role]);

    $pdo->prepare("
        UPDATE announcements
        SET remaining_spots = remaining_spots - 1
        WHERE id = ?
    ")->execute([$announcementId]);

    $pdo->commit();
    echo json_encode(['success' => true]);
} catch (Throwable $e) {
    $pdo->rollBack();
    http_response_code(500);
    echo json_encode(['error' => 'Join failed']);
}
```

```
}
```

Listato 2.24: join\_announcement.php: transazione completa

### 2.4.11 Match: dal join alla validazione, con stato pending/confirmed

La vista Match è una delle parti più importanti di GOAT, perché introduce un livello di controllo. Un'interazione tra due utenti non produce automaticamente un beneficio economico, cioè non c'è lo scambio di crediti, ma passa attraverso uno stato intermedio di attesa segnalato come **pending** nel database.

Questa scelta risolve un problema pratico, *come evitare che un utente simultaneamente partecipi a più match per ottenere crediti?* Il sistema impone che un soggetto terzo (admin) convalidi il match.

Il match nasce nel momento in cui un utente effettua la join a un annuncio. A livello di dominio, join significa:

- registrare la partecipazione con un ruolo derivato dal tipo annuncio,
- generare un codice di validazione,
- creare una riga nella tabella match per collegare creatore e partecipante,
- decrementare posti disponibili.

Queste operazioni non possono essere parziali: se fallisce una, devono fallire tutte. Per questo, join\_announcement.php usa una transazione.

```
$role = $announcement['type'] === 'richiesta' ? 'tutor' : 'student';
$code = 'GOAT-' . strtoupper(bin2hex(random_bytes(3)));

$pdo->beginTransaction();
try {
    $pdo->prepare("
        INSERT INTO announcement_participants
        (announcement_id, user_id, role, status, validation_code)
        VALUES (?, ?, ?, 'pending', ?)
    ")>execute([$announcementId, $userId, $role, $code]);

    $pdo->prepare("
        UPDATE announcements
        SET remaining_spots = remaining_spots - 1
    ")>execute();
}
```

```
        WHERE id = ? AND remaining_spots > 0
    ")->execute([$announcementId]);

    $pdo->prepare("
        INSERT INTO announcement_matches
        (announcement_id, creator_id, participant_id, validation_code,
         status, created_at)
        VALUES (?, ?, ?, ?, 'pending', NOW())
    ")->execute([$announcementId, (int)$announcement['created_by'],
        $userId, $code]);

    $pdo->commit();
    echo json_encode(['success' => true]);
} catch (Throwable $e) {
    $pdo->rollBack();
    http_response_code(500);
    echo json_encode(['error' => 'Join failed']);
}
```

Listato 2.25: join-announcement.php: inserimento partecipazione e generazione codice (backend)

L'aspetto rilevante è che la join scrive in due tabelle: `announcement_participants` (relazione utente–annuncio) e `announcement_matches` (relazione creatore–partecipante con stato e codice). Questo permette due letture diverse:

- la vista Annunci seguiti può basarsi sulla tabella partecipanti;
- la vista Match può basarsi sulla tabella match, più adatta a stati, validazione e storico.

La vista Match, lato frontend, è organizzata in due tab: `pending` e `confirmed`. Ogni elemento mostra un badge di stato e un'azione coerente allo stato in cui è.

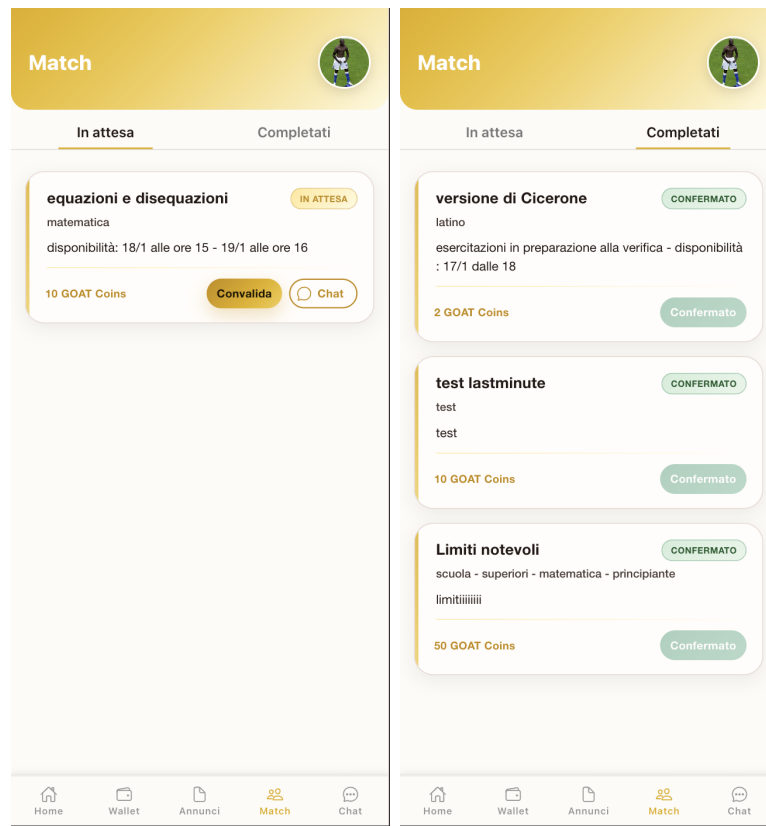


Figura 2.6: Screen della vista Match

```

activeTab: 'pending' | 'confirmed' = 'pending';
matches: Match[] = [];
loading = true;

get filtered(): Match[] {
  return this.matches.filter((m) => m.match_status ===
    this.activeTab);
}

switchTab(tab: 'pending' | 'confirmed') {
  this.activeTab = tab;
}

ngOnInit() {
  this.load();
}

load() {

```

```
this.loading = true;
this.service.getMyMatches().subscribe({
  next: (res) => {
    this.matches = res.data ?? [];
    this.loading = false;
  },
  error: () => {
    alert('Errore caricamento match');
    this.loading = false;
  },
});
}
```

Listato 2.26: MatchPage: filtro per stato e gestione tab (frontend)

Il servizio frontend esegue una semplice GET:

```
getMyMatches() {
  return this.http.get<{ success: boolean; data: Match[] }>(
    this.baseUrl + 'get_matches.php',
    { withCredentials: true }
  );
}
```

Listato 2.27: MatchService: recupero match utente (frontend)

Sul backend, `get_matches.php` deve includere match sia per il creatore sia per il partecipante: entrambi devono vedere il match in lista. Inoltre, la query deve evitare ambiguità sui parametri e deve garantire che vengano restituiti i campi realmente usati dal frontend cioè `match_status`, `validation_code`, `title`, `credits`, `data/ora evento`.

```
$userId = (int) $_SESSION['user']['id'];

$sql = "
SELECT
  m.id AS match_id,
  m.status AS match_status,
  m.validation_code,
  m.created_at,
  m.validated_at,

  a.id AS announcement_id,
  a.title,
  a.subtitle,
```

```
a.description,  
a.event_date,  
a.event_time,  
a.credits,  
a.created_by  
FROM announcement_matches m  
JOIN announcements a ON a.id = m.announcement_id  
WHERE  
    (m.creator_id = :uid OR m.participant_id = :uid)  
    AND a.is_visible = 1  
ORDER BY m.created_at DESC  
";  
  
$stmt = $pdo->prepare($sql);  
$stmt->execute([':uid' => $userId]);  
$rows = $stmt->fetchAll(PDO::FETCH_ASSOC);  
  
echo json_encode(['success' => true, 'data' => $rows]);
```

Listato 2.28: `get_matches.php`: match visibili a creatore e partecipante (backend)

A livello di UX, la vista Match presenta un'azione *Convalida* quando il match è pending. In questa fase, nel progetto è stato adottato un approccio volutamente semplice, il codice viene mostrato all'utente tramite alert e potrà essere comunicato/mostrato per la verifica, ma, lo stesso codice potrebbe essere rappresentato anche tramite QR.

```
showCode(match: Match) {  
    alert('Codice di convalida:\n${match.validation_code}');  
}
```

Listato 2.29: `MatchPage`: visualizzazione codice in pending (frontend)

### 2.4.12 Collegamento tra la vista Match e la dashboard amministrativa

Il punto di incontro principale tra l'interazione dell'utente e le operazioni di controllo svolte dall'admin tramite la dashboard è rappresentato dalla vista *Match*. È infatti la vista su cui si ha il processo di validazione.

Dal punto di vista del flusso applicativo, il match nasce nel momento in cui un utente partecipa a un annuncio pubblicato da un altro utente. In questa fase, il sistema registra una partecipazione e genera un match in stato

di attesa (**pending**). Tale stato è immediatamente visibile nella vista Match dell'applicazione mobile, sia per il creatore dell'annuncio sia per il partecipante, rendendo esplicito che l'interazione è stata avviata ma non ancora conclusa.

La dashboard admin interviene solo su questa entità. Attraverso il gestionale, l'admin visualizza l'elenco dei match in attesa e può procedere alla loro validazione. Questa operazione aggiorna lo stato del match da **pending** a **confirmed** e rappresenta il momento in cui il sistema riconosce formalmente l'avvenuto svolgimento dell'attività.

Dal punto di vista tecnico, la dashboard non modifica direttamente gli annunci né le partecipazioni, ma agisce sulla tabella dei match. L'aggiornamento dello stato del match viene quindi propagato in modo indiretto a tutte le viste che dipendono da tale informazione.

In particolare, la vista Match riflette immediatamente l'esito della validazione, mostrando il passaggio allo stato confermato (**confirmed**) e disabilitando le azioni associate ai match in attesa. Parallelamente, gli effetti della validazione si manifestano anche in altre parti dell'applicazione per esempio il Wallet dell'utente viene aggiornato con l'accredito dei crediti previsti.

Questa organizzazione rafforza la coerenza del modello concettuale di GOAT. L'utente non interagisce mai direttamente con la dashboard admin, ma vede in modo chiaro le conseguenze delle operazioni di validazione attraverso l'aggiornamento degli stati nella vista Match e sul Wallet. In tal modo, il sistema mantiene separati i livelli di interazione e di controllo, senza compromettere la trasparenza del funzionamento complessivo.

### 2.4.13 Chat tra utenti e continuità dopo il match

Nel progetto GOAT è presente anche una vista Chat, che completa il ciclo di interazione tra gli utenti. Se la Home e la vista Annunci permettono di creare l'incontro tra domanda e offerta, la Chat rappresenta invece lo spazio operativo in cui i due soggetti possono coordinarsi dopo la creazione del match.

Dal punto di vista del flusso applicativo, l'apertura della conversazione avviene a partire dalla vista Match. Il frontend richiama un endpoint che recupera una conversazione già esistente oppure ne crea una nuova associata al match selezionato. In questo modo, la comunicazione rimane agganciata a una relazione reale già presente nel dominio e non viene lasciata come canale generico scollegato dal resto del sistema.

```
openChat(m: Match) {
  this.chatService.getOrCreateConversation(m.match_id).subscribe({
    next: (id) => {
      if (id != null) this.router.navigate(['/chat', id]);
      else alert('Impossibile aprire la chat.');
```

Listato 2.30: MatchPage: apertura o creazione della conversazione (frontend)

La vista Chat mostra l'elenco delle conversazioni dell'utente, con l'altro partecipante, l'ultimo messaggio e l'eventuale numero di messaggi non letti. La lista viene recuperata tramite una chiamata HTTP autenticata e consente di trasformare il match da semplice stato amministrativo a relazione effettivamente utilizzabile dagli utenti.

```
getConversations(): Observable<ChatConversation[]> {
  return this.http
    .get<GetConversationsResponse>(this.baseUrl +
      'get_conversations.php', {
        withCredentials: true,
      })
    .pipe(
      map((res) => (res?.success && res?.data ? res.data : []))
    );
}
```

Listato 2.31: ChatService: recupero conversazioni utente (frontend)

All'interno della singola conversazione, il caricamento dei messaggi non avviene una sola volta, ma viene aggiornato periodicamente tramite polling. Questa scelta è coerente con l'impostazione generale del progetto, che privilegia semplicità implementativa e integrazione immediata con un backend PHP senza introdurre canali realtime più complessi come WebSocket.

```
interval(1000)
  .pipe(
    takeUntil(this.destroy$),
    switchMap(() => this.chatService.getMessages(cid)),
  )
  .subscribe((list) => {
    this.messages.set(list);
  });
```

Listato 2.32: ChatRoomPage: aggiornamento periodico dei messaggi (frontend)

Un ulteriore aspetto rilevante è la gestione dei messaggi non letti. Il servizio di chat mantiene un contatore globale, utilizzato poi nella barra di navigazione inferiore tramite un badge numerico. In questo modo, l'utente può percepire immediatamente la presenza di nuove comunicazioni anche quando si trova in altre viste dell'applicazione.

La Chat, quindi, non è un modulo accessorio, ma una prosecuzione naturale del match: prima il sistema mette in contatto due utenti, poi offre loro uno spazio persistente per organizzare concretamente l'attività.

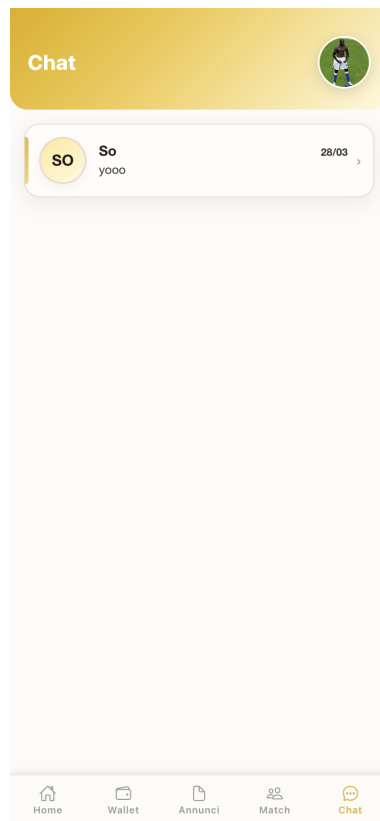


Figura 2.7: Screen della vista chat

#### 2.4.14 Wallet: saldo, riscatti e storico movimenti

Il Wallet rende esplicito un aspetto fondamentale di GOAT: ogni interazione valida produce un transazione di GOAT Coins e tale valore può essere speso

sia per lezioni che per voucher nei business locali. Il Wallet è stato progettato per mostrare saldo, voucher disponibili e tracciabilità tramite la schermata dei movimenti.

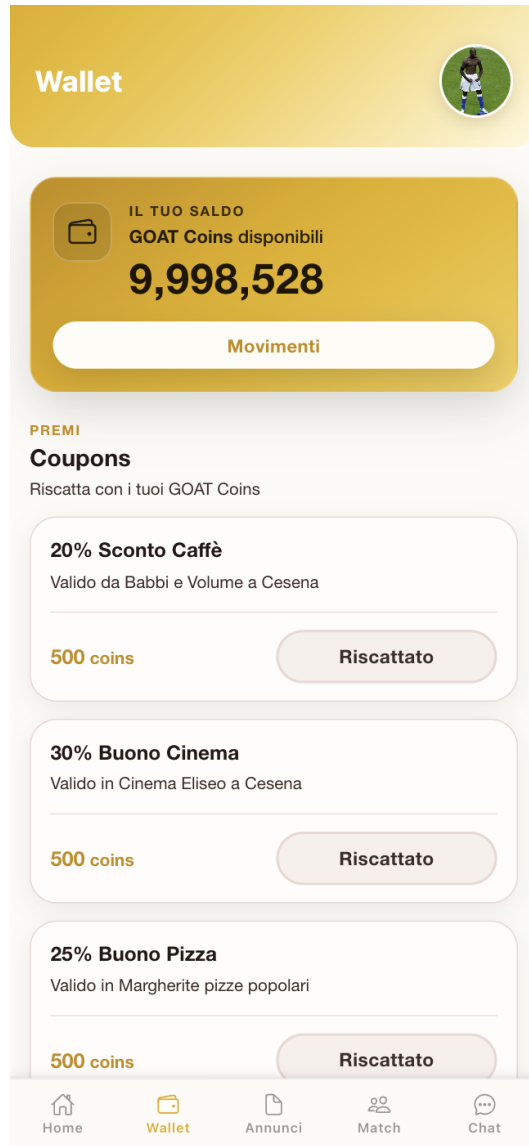


Figura 2.8: Screen vista Wallet

Il saldo è derivato dallo stato dell'utente autenticato richiamato nel `ngOnInit()` della vista.

```
ngOnInit() {  
  this.auth.user$.subscribe((user) => {
```

```

    this.user = user ?? undefined;
    this.credits = user?.goat_coins ?? 0;
  });

  if (!this.auth.currentUser) {
    this.auth.me();
  }
}

```

Listato 2.33: WalletPage: sincronizzazione saldo con AuthService (frontend)

Il riscatto di un coupon è un'operazione sensibile, deve essere atomica<sup>2</sup> e coerente, perché modifica sia saldo sia storico movimenti. Il frontend gestisce lo stato del pulsante e impedisce invii multipli con un flag `loadingCouponId`.

```

<button
  class="goat-btn-outline redeem-btn"
  [disabled]="credits < coupon.cost || coupon.redeemed ||
    loadingCouponId === coupon.id"
  (click)="redeemCoupon(coupon)"
>
  <span *ngIf="loadingCouponId !== coupon.id">
    {{ coupon.redeemed ? 'Riscattato' : 'Riscatta' }}
  </span>
  <span *ngIf="loadingCouponId === coupon.id" class="spinner"></span>
</button>

```

Listato 2.34: Wallet: pulsante riscatto con stati di disabilitazione (frontend)

L'operazione frontend invia una POST al backend includendo `withCredentials`, e al successo aggiorna la UI sottraendo i crediti.

```

async redeemCoupon(couponId: number): Promise<{ success: boolean;
  code: string }> {
  return await firstValueFrom(
    this.http.post<{ success: boolean; code: string }>(
      `${this.base}/redeem_coupon.php`,
      { coupon_id: couponId },
      { withCredentials: true }
    )
  );
}

```

<sup>2</sup>In informatica, "atomico" si riferisce a un'operazione o transazione indivisibile. Un'azione atomica viene eseguita interamente o per nulla, senza che altri processi possano interromperla o vederne gli stati intermedi.

Listato 2.35: WalletService: redeemCoupon con sessione (frontend)

Sul backend, il riscatto deve verificare l'autenticazione, verificare saldo sufficiente, aggiornare la tabella `users` e inserire una riga in `wallet_redemptions`.

```
$pdo->beginTransaction();
try {
    $stmt = $pdo->prepare("SELECT goat_coins FROM users WHERE id = ?
        FOR UPDATE");
    $stmt->execute([$userId]);
    $user = $stmt->fetch(PDO::FETCH_ASSOC);

    if (!$user || (int)$user['goat_coins'] < (int)$coupon['cost']) {
        throw new Exception('Not enough credits');
    }

    $pdo->prepare("UPDATE users SET goat_coins = goat_coins - ? WHERE
        id = ?")
        ->execute([(int)$coupon['cost'], $userId]);

    $pdo->prepare("
        INSERT INTO wallet_redemptions (user_id, coupon_key, cost,
            redeemed_at)
        VALUES (?, ?, ?, NOW())
    ")>execute([$userId, $couponKey, (int)$coupon['cost']]);

    $pdo->commit();
    echo json_encode(['success' => true, 'code' => $coupon['code']]);
} catch (Throwable $e) {
    $pdo->rollBack();
    http_response_code(400);
    echo json_encode(['error' => 'Redeem failed']);
}
```

Listato 2.36: redeem\_coupon.php: riscatto transazionale e log movimenti (backend)

Infine, lo storico movimenti è ottenuto con un endpoint che legge `wallet_redemptions` e mappa le chiavi coupon in etichette. Questo storico non è un semplice “diario”, è ciò che permette alla UI di indicare “già riscattato” senza richiedere all’utente di aprire manualmente i movimenti. In altre parole, la coerenza della UI dipende dalla persistenza corretta dei dati dei movimenti.

### 2.4.15 Profilo utente, modifica dati e cronologia personale

Oltre alle viste strettamente operative, il progetto include anche una vista Profilo che raccoglie in un unico punto le informazioni identificative e una sintesi dell'attività dell'utente. Questa pagina non ha un ruolo puramente estetico, ma costituisce una forma di dashboard personale: mostra nome, email, ruolo, materia principale, livello raggiunto, crediti accumulati e numero di lezioni svolte come tutor o come studente.

Il caricamento iniziale del profilo si basa ancora una volta sul metodo `me()` del servizio di autenticazione. In questo modo, la vista profilo non introduce una fonte alternativa di verità, ma riutilizza lo stesso stato utente condiviso dall'intera applicazione.

```
async loadProfile() {
  const res = await this.auth.me();
  const u = res.user;

  this.userName = u.name;
  this.userEmail = u.email || '';
  this.userRole = this.mapRole(u.user_role);
  this.mainSubject = u.main_subject || '';
  this.level = u.level;
  this.goatCoins = u.goat_coins;
  this.lessonsAsTutor = u.lessons_as_tutor;
  this.lessonsAsStudent = u.lessons_as_student;
}
```

Listato 2.37: ProfilePage: caricamento dati profilo (frontend)

Accanto alla visualizzazione, il progetto prevede anche una vista separata di modifica profilo. Qui l'utente può aggiornare alcuni dati personali come nome, materia principale e avatar. Anche in questo caso il flusso è coerente con il resto dell'architettura: la modifica avviene tramite endpoint backend e, al successo, il servizio di autenticazione aggiorna il dato locale così da riflettere immediatamente il cambiamento nelle altre viste.

```
async save() {
  await this.auth.updateProfile({
    name: this.name,
    main_subject: this.main_subject,
    avatar_url: this.avatar_url,
  });
}
```

```
this.router.navigateByUrl('/profile', { replaceUrl: true });
}
```

Listato 2.38: ProfileEditPage: aggiornamento del profilo (frontend)

Un elemento particolarmente interessante della vista Profilo è la cronologia personale delle attività. Il frontend non si limita a mostrare dati anagrafici, ma costruisce una lista unica combinando movimenti del Wallet e attività di lezione. Questo consente all'utente di avere una lettura sintetica ma ordinata del proprio percorso sulla piattaforma, mantenendo insieme dimensione economica e dimensione formativa.

La presenza di questa vista rafforza la tracciabilità generale del sistema: non solo il backend registra gli eventi, ma il frontend li riorganizza in una forma leggibile e centrata sull'esperienza personale dell'utente.

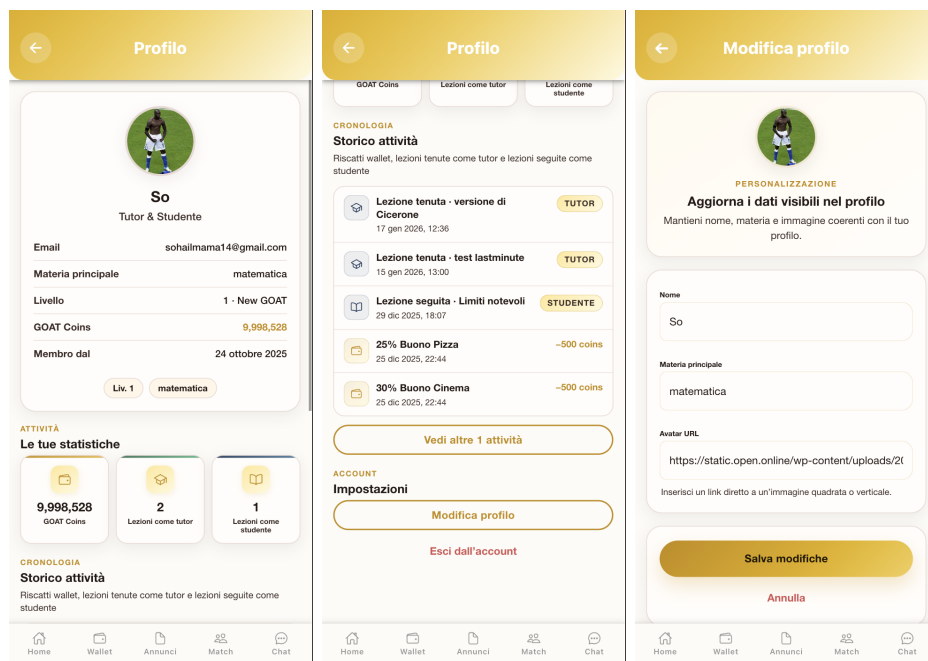


Figura 2.9: Screen vista Profilo

#### 2.4.16 Validazione e trasferimento crediti: completamento del ciclo

Il completamento del match richiede un'azione esterna (da parte della dashboard admin) che aggiorni lo stato a **confirmed**. Il punto fondamentale è

che la validazione non è solo un aggiornamento di stato, ma, è l'evento che sblocca il trasferimento dei crediti.

Nel backend, `validate_match.php` deve:

1. verificare che il match esista e sia `pending`;
2. aggiornare lo stato a `confirmed` e salvare `validated_at` che contiene il timestamp di quando è stato validato il match;
3. trasferire i crediti in modo coerente cioè ci deve essere accreditato al partecipante o a chi deve riceverli.

```
$input = json_decode(file_get_contents('php://input'), true);
$matchId = (int) ($input['match_id'] ?? 0);

if ($matchId <= 0) {
    http_response_code(400);
    echo json_encode(['error' => 'match_id missing']);
    exit;
}

$pdo->beginTransaction();
try {
    $stmt = $pdo->prepare("
        SELECT
            m.id, m.status, m.creator_id, m.participant_id,
            m.announcement_id,
            a.credits
        FROM announcement_matches m
        JOIN announcements a ON a.id = m.announcement_id
        WHERE m.id = ?
        FOR UPDATE
    ");
    $stmt->execute([$matchId]);
    $m = $stmt->fetch(PDO::FETCH_ASSOC);

    if (!$m) {
        throw new Exception('Match not found');
    }
    if ($m['status'] !== 'pending') {
        throw new Exception('Match not pending');
    }

    $upd = $pdo->prepare("

```

```
        UPDATE announcement_matches
        SET status = 'confirmed', validated_at = NOW()
        WHERE id = ?
    );
    $upd->execute([$matchId]);

    $credits = (int) $m['credits'];

    // Esempio: accredito al partecipante
    $pdo->prepare("
        UPDATE users
        SET goat_coins = goat_coins + ?
        WHERE id = ?
    ")>execute([$credits, (int)$m['participant_id']]);

    $pdo->commit();
    echo json_encode(['success' => true]);
} catch (Throwable $e) {
    $pdo->rollBack();
    http_response_code(400);
    echo json_encode(['error' => 'Validation failed']);
}
```

Listato 2.39: validate\_match.php: conferma match e trasferimento crediti (backend)

*Questo passaggio chiude il cerchio: un utente scopre un annuncio in Home, partecipa (join), genera un match in stato pending, il match viene validato nel gestionale dopo l'incontro e infine il saldo Wallet riflette l'esito della collaborazione.*

### 2.4.17 Gestione dello storico e tracciabilità delle interazioni

Uno degli obiettivi del progetto GOAT è la tracciabilità completa delle interazioni. Ogni azione rilevante viene registrata in modo persistente, consentendo:

- ricostruzione dello storico dell'utente;
- verifica delle collaborazioni avvenute;
- analisi futura del funzionamento del sistema.

La tabella `announcement_matches` rappresenta il cuore dello storico delle collaborazioni. Ogni riga collega due utenti a un annuncio e ne registra lo stato.

```
CREATE TABLE announcement_matches (  
  id INT AUTO_INCREMENT PRIMARY KEY,  
  announcement_id INT NOT NULL,  
  creator_id INT NOT NULL,  
  participant_id INT NOT NULL,  
  validation_code VARCHAR(20),  
  status ENUM('pending','confirmed'),  
  created_at DATETIME,  
  validated_at DATETIME  
);
```

Listato 2.40: Struttura di `announcement_matches`

```
SELECT  
  m.id,  
  m.status,  
  m.validation_code,  
  a.title,  
  a.credits,  
  m.created_at,  
  m.validated_at  
FROM announcement_matches m  
JOIN announcements a ON a.id = m.announcement_id  
WHERE m.creator_id = :uid OR m.participant_id = :uid  
ORDER BY m.created_at DESC;
```

Listato 2.41: `get_matches.php`: storico match utente

## 2.5 Accessibilità nell'implementazione

Un aspetto trasversale del progetto riguarda l'attenzione all'accessibilità, intesa come la capacità del sistema di essere fruibile in modo efficace dal maggior numero possibile di utenti, indipendentemente dal dispositivo utilizzato o dalle competenze tecniche pregresse.

Dal punto di vista implementativo, l'adozione di Angular e Ionic ha costituito una base favorevole in questo senso. I componenti Ionic, come `ion-button`, `ion-label`, `ion-tab-button` e `ion-input`, generano automaticamente markup HTML semantico, garantendo attributi `role` e `aria-label` corretti per

i lettori di schermo. Questo riduce significativamente il rischio di produrre interfacce visivamente corrette ma non navigabili da utenti che fanno uso di tecnologie assistive.

La navigazione tramite bottom bar, implementata con `ion-tab-bar` e `ion-tab-button`, rispetta le convenzioni di accessibilità dei dispositivi mobili, consentendo la navigazione tra le sezioni principali anche attraverso comandi vocali o tastiera. La stessa struttura viene nascosta nelle schermate di autenticazione e nella vista della singola conversazione di chat, riducendo il carico cognitivo dell'utente e limitando gli elementi interattivi a quelli strettamente necessari al contesto.

Un ulteriore elemento rilevante riguarda la gestione degli stati interattivi nei form e nei pulsanti. In più punti dell'applicazione, i pulsanti vengono disabilitati durante le operazioni in corso tramite la direttiva `[disabled]`, e il loro testo cambia per riflettere lo stato dell'operazione, ad esempio da "Pubblica annuncio" a "Salvataggio...". Questo meccanismo non ha solo una funzione di prevenzione degli errori, ma fornisce anche un feedback esplicito all'utente, riducendo l'ambiguità sullo stato del sistema in un momento in cui l'interfaccia non risponde agli input.

Sul piano del contrasto e della leggibilità, l'utilizzo dei temi predefiniti di Ionic garantisce una base di partenza conforme alle linee guida WCAG<sup>3</sup> per quanto riguarda il rapporto tra colore del testo e sfondo. Le personalizzazioni grafiche introdotte non alterano questa impostazione di base.

Infine, la presenza di feedback visivi coerenti, come badge numerici per i messaggi non letti, indicatori di caricamento e messaggi di errore espliciti, contribuisce a rendere il sistema comprensibile e navigabile anche per utenti meno esperti, allineando l'accessibilità tecnica con quella percepita.

## 2.6 Dashboard amministrativa come sottosistema di controllo

La dashboard amministrativa rappresenta un sottosistema separato dall'applicazione principale, ma strettamente integrato a livello di dati. Il suo scopo è fornire un punto di controllo centrale per la validazione dei match.

### 2.6.1 Autenticazione amministratore

L'accesso alla dashboard è protetto da un login dedicato, basato su credenziali hard-coded per semplicità, data la natura prototipale del progetto.

---

<sup>3</sup>Web Content Accessibility Guidelines: insieme di linee guida internazionali per l'accessibilità dei contenuti web.

```
if (  
    $_POST['email'] === 'admin@goat.it' &&  
    $_POST['password'] === 'passwordadmin26'  
) {  
    $_SESSION['admin'] = true;  
    header('Location: dashboard.php');  
    exit;  
}
```

Listato 2.42: admin/index.php: login amministratore

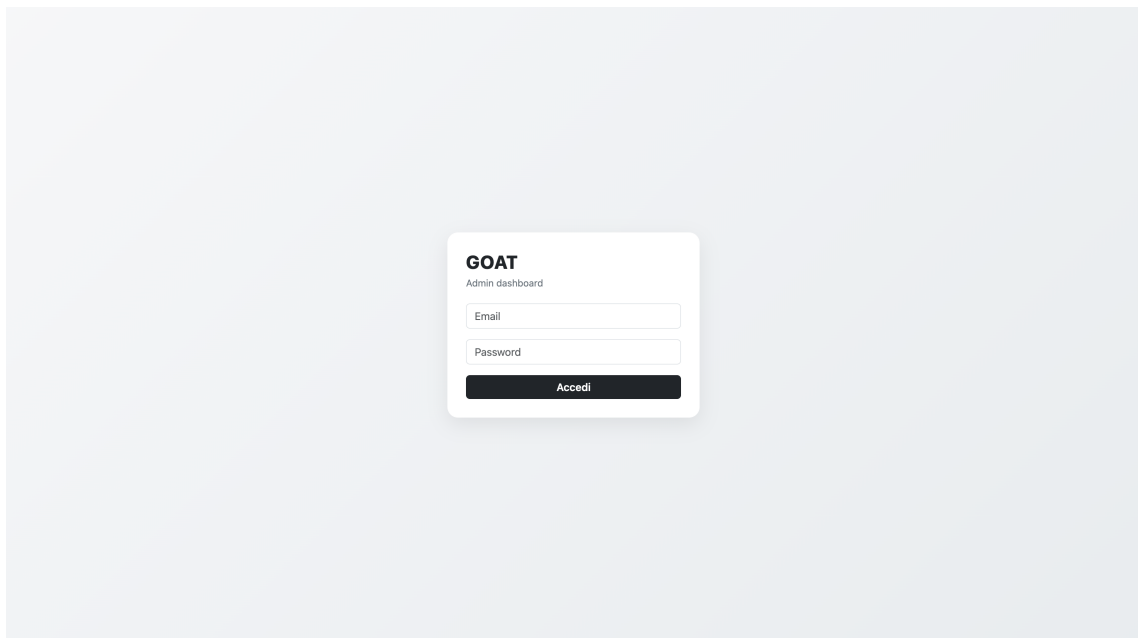


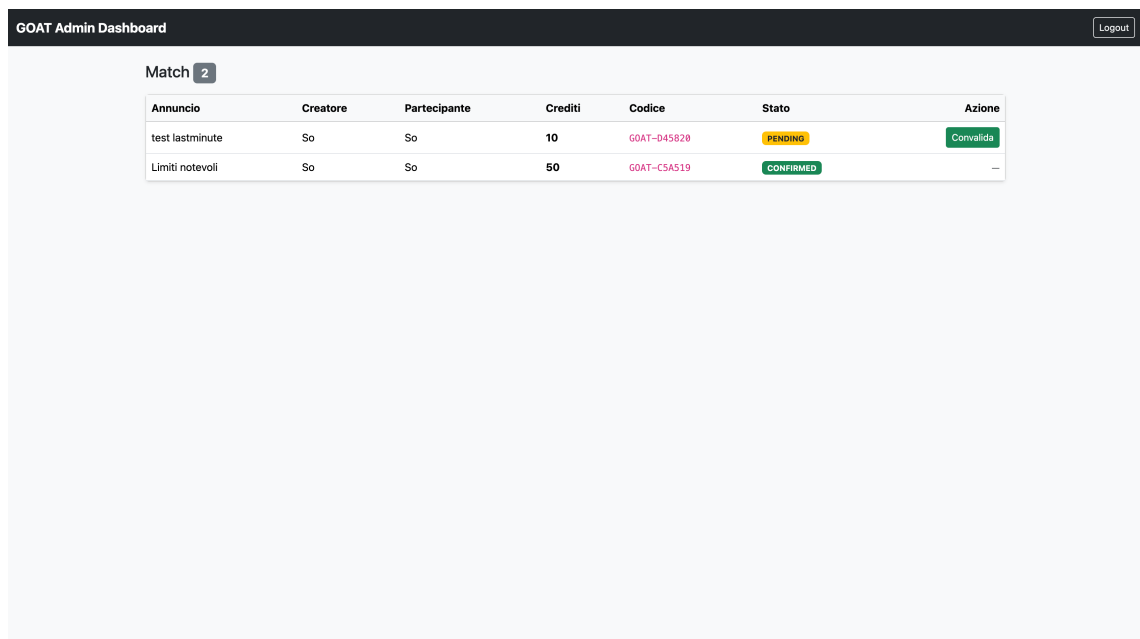
Figura 2.10: Login della pagina admin

## 2.6.2 Visualizzazione dei match

```
SELECT  
    m.id,  
    m.status,  
    m.validation_code,  
    a.title,  
    a.credits,  
    u1.name AS creator,  
    u2.name AS participant
```

```
FROM announcement_matches m
JOIN announcements a ON a.id = m.announcement_id
JOIN users u1 ON u1.id = m.creator_id
JOIN users u2 ON u2.id = m.participant_id;
```

Listato 2.43: dashboard.php: elenco match



The screenshot shows the 'GOAT Admin Dashboard' with a 'Logout' button in the top right. Below the header, there is a section titled 'Match' with a tab indicator '2'. It contains a table with the following data:

| Annuncio        | Creatore | Partecipante | Crediti | Codice      | Stato     | Azione    |
|-----------------|----------|--------------|---------|-------------|-----------|-----------|
| test lastminute | So       | So           | 10      | GOAT-045828 | PENDING   | Convalida |
| Limiti notevoli | So       | So           | 50      | GOAT-C5A519 | CONFIRMED | —         |

Figura 2.11: Vista della home dell'admin

### 2.6.3 Validazione del match

```
UPDATE announcement_matches
SET status = 'confirmed',
    validated_at = NOW()
WHERE id = ?;
```

Listato 2.44: validate\_match.php: conferma match

Questa azione completa il ciclo informativo e abilita il trasferimento dei crediti.

## 2.7 Aspetti implementativi aggiuntivi

Oltre al flusso principale già descritto, ci sono aspetti implementativi che sono importanti per comprendere il funzionamento reale del sistema. Si tratta di elementi meno visibili dal punto di vista dell'interfaccia, ma essenziali per la robustezza dell'architettura: la configurazione delle API, la gestione estesa del profilo utente, il backend completo della chat e la chiusura corretta della sessione.

### 2.7.1 Configurazione delle API di CORS e della sessione PHP

Un aspetto fondamentale dell'implementazione riguarda il fatto che il frontend Ionic/Angular e il backend PHP operano su origini differenti durante lo sviluppo. Per rendere possibile l'invio del cookie di sessione, il backend non si limita ad accettare richieste HTTP generiche, ma configura esplicitamente le intestazioni CORS e i parametri del cookie di sessione.

Nel file `cors.php`, il backend consente le richieste solo da un insieme definito di origini locali e abilita l'uso delle credenziali:

```
$origin = $_SERVER['HTTP_ORIGIN'] ?? '';  
  
$allowed = [  
    'http://localhost:4200',  
    'http://127.0.0.1:4200',  
    'http://localhost:8100',  
    'http://127.0.0.1:8100',  
];  
  
if (in_array($origin, $allowed, true)) {  
    header("Access-Control-Allow-Origin: $origin");  
    header('Access-Control-Allow-Credentials: true');  
    header('Vary: Origin');  
}
```

Listato 2.45: `cors.php`: abilitazione controllata delle richieste cross-origin

Questa scelta è coerente con l'uso di `withCredentials` nel frontend: non basta che il client voglia inviare il cookie, il server deve anche autorizzarlo esplicitamente.

In parallelo, il file `session.php` definisce in modo centralizzato il comportamento della sessione PHP:

```
session_set_cookie_params([
    'lifetime' => 60 * 60 * 24 * 7,
    'path' => '/',
    'domain' => 'localhost',
    'secure' => false,
    'httponly' => true,
    'samesite' => 'Lax'
]);

session_start();
```

Listato 2.46: `session.php`: parametri condivisi del cookie di sessione

Dal punto di vista architetturale, questa soluzione evita duplicazioni e rende uniforme il comportamento di tutti gli endpoint autenticati. Ogni API che include `session.php` eredita lo stesso modello di sessione, migliorando la prevedibilità del sistema.

### 2.7.2 Recupero esteso e aggiornamento del profilo utente

Nel progetto, l'endpoint `me.php` non restituisce solo i dati minimi della sessione, ma interroga il database per ottenere anche informazioni estese sul profilo dell'utente. Questa scelta permette al frontend di popolare viste come Profilo e Wallet senza dover frammentare il caricamento in più chiamate separate.

```
$stmt = $pdo->prepare("
    SELECT
        id,
        name,
        email,
        user_role,
        main_subject,
        level,
        goat_coins,
        lessons_as_tutor,
        lessons_as_student,
        avatar_url,
        created_at,
```

```
        updated_at
    FROM users WHERE id = ? LIMIT 1
");
$stmt->execute([$userId]);
$user = $stmt->fetch();

echo json_encode(['user' => $user]);
```

Listato 2.47: me.php: recupero esteso del profilo utente dal database

Questa implementazione è interessante perché separa due livelli: la sessione conserva i dati minimi necessari a riconoscere l'utente autenticato, mentre `me.php` ricostruisce il profilo applicativo aggiornato leggendo lo stato persistente nel database.

Accanto al recupero dei dati è presente anche l'endpoint `update_profile.php`, che consente all'utente di aggiornare alcune informazioni personali. Anche in questo caso il backend mantiene il controllo sui vincoli minimi, per esempio impedendo un salvataggio con nome vuoto.

```
$name = trim($data['name'] ?? '');
$mainSubject = trim($data['main_subject'] ?? '');
$avatarUrl = trim($data['avatar_url'] ?? '');
$bio = trim($data['bio'] ?? '');

if ($name === '') {
    http_response_code(422);
    echo json_encode(['error' => 'Name is required']);
    exit;
}

$stmt = $pdo->prepare("
    UPDATE users
    SET name = ?, main_subject = ?, avatar_url = ?, bio = ?,
        updated_at = NOW()
    WHERE id = ?
");
```

Listato 2.48: update\_profile.php: aggiornamento dati del profilo

Dal punto di vista progettuale, questa parte completa la logica del profilo: il sistema non si limita a mostrare dati dell'utente, ma permette anche di mantenerli coerenti e aggiornati nel tempo.

### 2.7.3 Implementazione backend della chat

Nel capitolo è già stata discussa la chat come prosecuzione naturale del match. Analizzando il codice backend, emerge però una struttura più articolata, basata su endpoint distinti per creazione conversazione, recupero metadati, lettura messaggi, invio e marcatura dei messaggi come letti.

Il primo passaggio è l'apertura o creazione della conversazione a partire dal match. L'endpoint `get_or_create_conversation.php` verifica anzitutto che l'utente corrente appartenga davvero al match, evitando l'accesso a conversazioni non pertinenti.

```
$mStmt = $pdo->prepare("
    SELECT id FROM announcement_matches
    WHERE id = :mid AND (creator_id = :uid OR participant_id = :uid2)
    LIMIT 1
");
$mStmt->execute([':mid' => $matchId, ':uid' => $userId, ':uid2' =>
    $userId]);

$cStmt = $pdo->prepare("SELECT id FROM conversations WHERE match_id
    = ? LIMIT 1");
$cStmt->execute([$matchId]);
```

Listato 2.49: `get_or_create_conversation.php`: verifica accesso e creazione conversazione

Una volta ottenuto l'identificativo della conversazione, il backend mette a disposizione due letture differenti. La prima, `get_conversation.php`, restituisce i metadati della conversazione e l'identità dell'altro utente; la seconda, `get_messages.php`, restituisce invece il contenuto ordinato cronologicamente dei messaggi.

```
$stmt = $pdo->prepare("
    SELECT id, conversation_id, sender_id, text, created_at, read_at
    FROM messages
    WHERE conversation_id = :cid
    ORDER BY created_at ASC
");
$stmt->execute([':cid' => $conversationId]);
```

Listato 2.50: `get_messages.php`: recupero ordinato dei messaggi della conversazione

Il fatto che i messaggi vengano recuperati in ordine crescente da `created_at` è coerente con l'uso di polling lato frontend: ogni aggiornamento ricostruisce la sequenza completa della conversazione in modo stabile e prevedibile.

L'invio del messaggio è gestito da `send_message.php`, che verifica nuovamente l'appartenenza dell'utente alla conversazione prima di inserire il testo nella tabella `messages`.

```
$ins = $pdo->prepare("
    INSERT INTO messages (conversation_id, sender_id, text)
    VALUES (:cid, :uid, :text)
");
$ins->execute([
    ':cid' => $conversationId,
    ':uid' => $userId,
    ':text' => $text,
]);
```

Listato 2.51: `send_message.php`: inserimento di un nuovo messaggio

Un ulteriore aspetto rilevante riguarda la gestione dei messaggi letti e non letti. Nel progetto sono presenti due endpoint distinti: uno per segnare come letti i messaggi di una singola conversazione, e uno per azzerare globalmente tutti i non letti dell'utente.

```
$upd = $pdo->prepare("
    UPDATE messages
    SET read_at = NOW()
    WHERE conversation_id = :cid AND sender_id != :uid AND read_at IS
        NULL
");
$upd->execute([':cid' => $conversationId, ':uid' => $userId]);
```

Listato 2.52: `mark_conversation_read.php`: aggiornamento dello stato di lettura

## 2.7.4 Logout e chiusura esplicita della sessione

Nel flusso di autenticazione un aspetto spesso trascurato è l'uscita corretta dall'applicazione. Nel progetto, `logout.php` non si limita a cancellare la variabile `$_SESSION`, ma elimina anche il cookie di sessione lato client e distrugge la sessione corrente.

```
$_SESSION = [];  
  
if (ini_get('session.use_cookies')) {  
    $params = session_get_cookie_params();  
    setcookie(  
        session_name(),  
        '',  
        [  
            'expires' => time() - 42000,  
            'path' => $params['path'],  
            'domain' => $params['domain'],  
            'secure' => $params['secure'],  
            'httponly' => $params['httponly'],  
            'samesite' => $params['samesite'] ?? 'Lax'  
        ],  
        0,  
    );  
}  
  
session_destroy();
```

Listato 2.53: logout.php: invalidazione completa della sessione

Dal punto di vista della sicurezza applicativa, questo comportamento riduce il rischio di sessioni residue e rende esplicita la chiusura dello stato autenticato. Anche questa parte contribuisce alla coerenza complessiva del sistema, perché garantisce che la sessione utente venga trattata come una risorsa da inizializzare, usare e invalidare in modo controllato.

# Capitolo 3

## Validazione del progetto

La fase di validazione rappresenta un momento centrale nel ciclo di vita di un sistema software, poiché consente di verificare se la soluzione progettata e implementata risponde in modo coerente agli obiettivi individuati nella fase di analisi. Nel caso del progetto GOAT, la validazione non si limita a un controllo tecnico del corretto funzionamento dell'applicazione, ma coinvolge anche aspetti concettuali, educativi e organizzativi, in quanto il sistema nasce per rispondere a una criticità reale osservata nel contesto territoriale.

Nel Capitolo 1 è stato analizzato il problema di partenza: l'assenza di uno strumento strutturato capace di coordinare e formalizzare le interazioni di peer tutoring tra studenti, garantendo tracciabilità, equità e riconoscimento del valore delle attività svolte. Nel Capitolo 2 è stata descritta la traduzione di tali esigenze in una soluzione software concreta, basata su un'architettura client-server, su una modellazione esplicita delle entità coinvolte e su un ciclo di vita ben definito delle collaborazioni. La validazione consiste dunque nel verificare la coerenza tra questi tre livelli: analisi del problema, progettazione concettuale e implementazione tecnica.

### 3.1 Validazione funzionale

Dal punto di vista funzionale, il sistema soddisfa i requisiti individuati nella fase di analisi. L'utente può pubblicare annunci, distinguendo tra richieste e offerte di competenze, e tali annunci costituiscono l'avvio formale di un processo collaborativo. La partecipazione non avviene in modo informale o implicito, ma attraverso una procedura controllata che verifica vincoli di integrità quali la disponibilità dei posti, la non partecipazione ai propri annunci e l'assenza di duplicazioni. Questa scelta progettuale consente di superare una delle criticità iniziali, ovvero la difficoltà nel distinguere tra accordi presi e attività effettiva-

mente svolte.

Un elemento centrale della validazione riguarda il modello del *match*. La decisione di separare concettualmente la semplice partecipazione dall'effettiva conclusione della collaborazione introduce uno stato intermedio (*pending*) che impedisce al sistema di attribuire automaticamente valore a un'interazione non ancora verificata. Tale meccanismo risponde direttamente al problema della mancanza di controllo nei contesti informali: finché il match non viene validato, non produce effetti economici all'interno della piattaforma. Solo a seguito della conferma da parte dell'amministratore il sistema procede al trasferimento dei crediti, aggiornando in modo coerente lo stato della collaborazione e il saldo degli utenti coinvolti.

## 3.2 Validazione tecnica

Dal punto di vista tecnico, la validazione si concentra sulla robustezza delle operazioni critiche. Le funzionalità di join, validazione del match e riscatto dei voucher sono state implementate mediante transazioni sul database. L'utilizzo esplicito di *beginTransaction()*, dei blocchi *FOR UPDATE* e del meccanismo di rollback in caso di errore garantisce la consistenza dei dati anche in presenza di operazioni concorrenti. Questo aspetto risulta particolarmente rilevante in un sistema che gestisce crediti virtuali, poiché eventuali incoerenze potrebbero compromettere la fiducia degli utenti nella piattaforma.

Un ulteriore elemento di validazione riguarda la tracciabilità delle interazioni. Ogni match viene registrato in modo persistente, associando creatore, partecipante, stato, codice di validazione e timestamp di creazione e conferma. Tale scelta non ha solo una funzione tecnica, ma costituisce anche un presupposto per la trasparenza del sistema. La possibilità di ricostruire a posteriori lo storico delle collaborazioni rafforza la credibilità della piattaforma e consente, in prospettiva, eventuali analisi statistiche sull'impatto del progetto.

## 3.3 Limiti del prototipo e validazione educativa

La validazione non può tuttavia prescindere da una riflessione sui limiti del prototipo sviluppato. Alcune scelte, come l'autenticazione amministrativa semplificata o l'assenza di un sistema reputazionale tra utenti, derivano dalla natura prototipale del progetto e non rappresentano soluzioni definitive per

un'eventuale distribuzione. Allo stesso modo, il modello dei crediti è stato implementato in forma essenziale, privilegiando la coerenza del flusso rispetto alla complessità di scenari economici più articolati. Tali limiti non compromettono la validità del modello, ma evidenziano possibili direzioni di sviluppo futuro.

Oltre alla validazione tecnica, è opportuno considerare anche la dimensione educativa del progetto. GOAT non si configura come una semplice piattaforma di scambio, bensì come un ambiente regolato in cui la collaborazione tra pari viene formalizzata, riconosciuta e incentivata. Il sistema dei crediti non assume la funzione di remunerazione economica tradizionale, ma opera come meccanismo simbolico di responsabilizzazione e reciprocità. In questo senso, la validazione del progetto riguarda anche la sua coerenza rispetto agli obiettivi pedagogici iniziali: promuovere inclusione, partecipazione attiva e valorizzazione delle competenze individuali.

### **3.4 Validazione sul campo: focus group nelle scuole**

Al fine di integrare la valutazione tecnica del sistema descritto nei paragrafi precedenti, è stata condotta una fase di validazione sul campo, con l'obiettivo di osservare il comportamento degli utenti in un contesto reale e raccogliere dati sulla percezione del progetto. Tale fase assume un ruolo centrale nel processo di sviluppo, in quanto consente di verificare la coerenza tra le scelte progettuali illustrate nel Capitolo 2 e le effettive esigenze degli utenti finali. Il focus group è stato condotto all'interno delle principali scuole di Cesena, sia tecnici che professionali, coinvolgendo studenti appartenenti a diverse classi. La scelta di questo ambiente è motivata dalla natura stessa del progetto GOAT.

Durante gli incontri, è stata effettuata una presentazione strutturata dell'applicazione, comprendente una descrizione delle funzionalità principali, del flusso di interazione e del modello concettuale alla base del sistema, con particolare attenzione al meccanismo di scambio fondato sulle competenze e al sistema di crediti. Al termine della presentazione, agli studenti è stato somministrato un questionario, progettato per raccogliere valutazioni e osservazioni sull'aspetto estetico che funzionale dell'applicazione. Questo approccio ha permesso di ottenere una visione articolata dell'esperienza utente, combinando dati sintetici con considerazioni più approfondite.

### 3.4.1 Analisi dei risultati del focus group

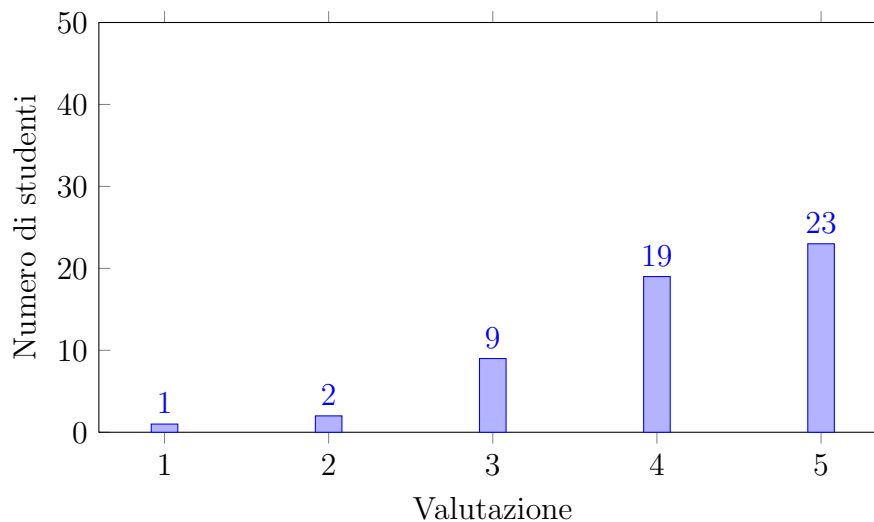
Il questionario digitale somministrato ai partecipanti del focus group aveva l'obiettivo di valutare la comprensione delle funzionalità dell'applicazione e di raccogliere eventuali feedback concreti previa visualizzazione di tutti gli aspetti del progetto.

Il campione utilizzato è stato di convenienza, in quanto composto da studenti selezionati sulla base della disponibilità e del contesto scolastico coinvolto nel focus group. Questa modalità di selezione è coerente con l'impostazione qualitativa del focus group, che privilegia la profondità delle osservazioni rispetto alla rappresentatività statistica del campione.

La composizione del questionario era di 18 domande di cui 14 a risposta chiusa con una scala di valutazione da 1 a 5 in base al grado di apprezzamento rispetto agli aspetti analizzati, mentre le restanti 4 domande aperte erano finalizzate a raccogliere suggerimenti, osservazioni e proposte di miglioramento relative alle funzionalità implementative dell'applicazione, essendo loro i consumatori finali del prodotto.

#### Comprensione e facilità d'uso

Dai risultati emerge che la maggior parte degli studenti ha capito facilmente a cosa serve l'app già al primo utilizzo.

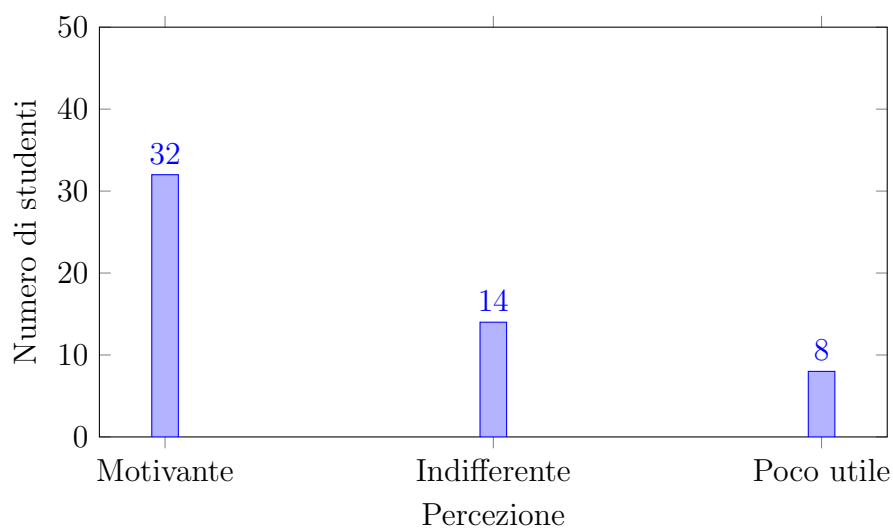


Come si può osservare dal grafico, la maggior parte delle valutazioni si concentra tra 4 e 5, indicando una buona comprensione generale dell'applicazione.

Nonostante questo, alcuni studenti hanno segnalato difficoltà legate soprattutto alla gestione delle richieste e alla chiarezza dell'interfaccia.

### Sistema dei crediti

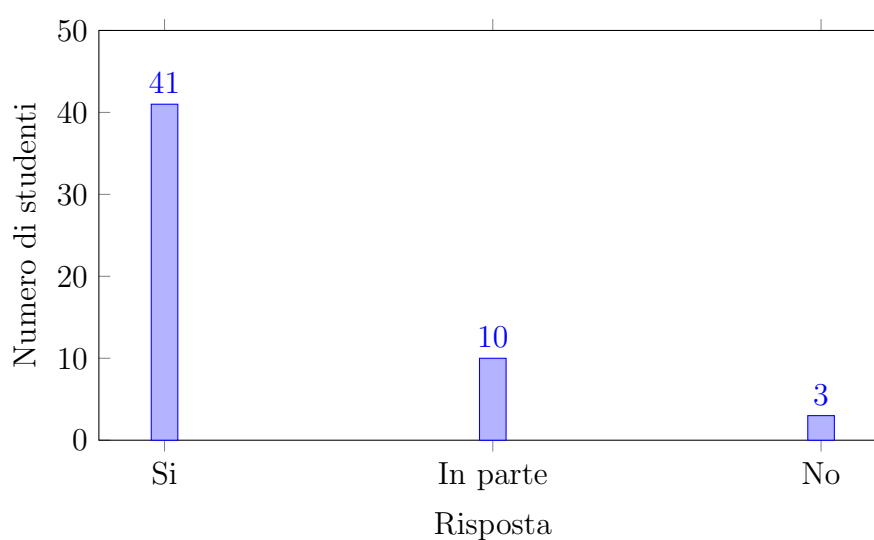
Il sistema dei GOAT Coins è stato generalmente percepito come positivo.



La maggior parte degli studenti considera il sistema motivante, ma una parte significativa lo percepisce come poco chiaro o poco utile.

### Utilità del progetto

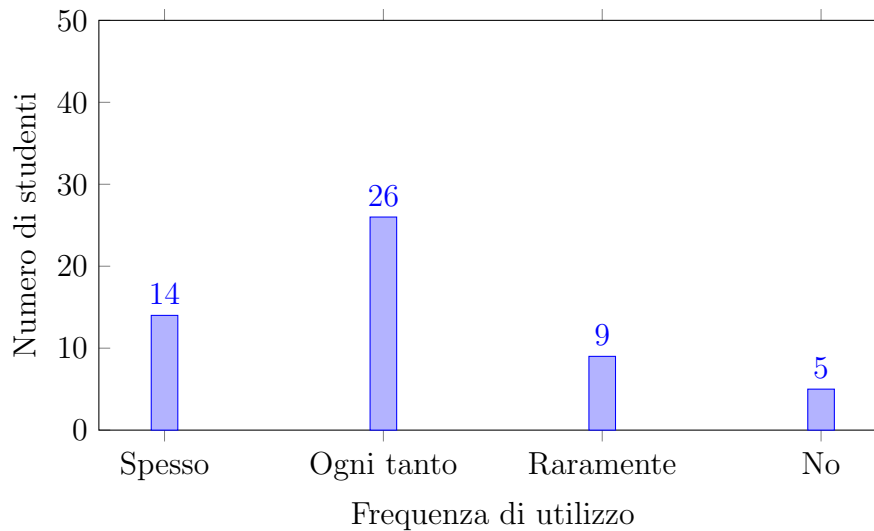
Un altro aspetto analizzato riguarda l'utilità percepita del progetto.



Dal grafico si nota chiaramente che la maggioranza degli studenti ritiene il progetto utile nella quotidianità scolastica.

### Disponibilità all'utilizzo

È stato inoltre analizzato quanto gli studenti sarebbero disposti a utilizzare il servizio.



La maggior parte degli studenti ha dichiarato che utilizzerebbe l'app almeno occasionalmente, segno che il progetto ha un buon potenziale di adozione.

### 3.4.2 Comprensione del modello e usabilità percepita

L'analisi dei dati raccolti evidenzia come il modello generale dell'applicazione sia stato compreso dalla maggior parte degli studenti. Soprattutto, la distinzione tra le due modalità principali di utilizzo, richiesta di supporto e offerta di tutoraggio, è risultata chiara e coerente con le aspettative degli utenti. Nonostante ciò, una parte degli studenti ha segnalato alcune difficoltà nella comprensione immediata dell'interfaccia e delle dinamiche operative. Questo elemento suggerisce che, sebbene la struttura concettuale del sistema sia solida, l'esperienza utente potrebbe essere ulteriormente ottimizzata. In particolare, emerge la necessità di introdurre meccanismi di onboarding più esplicativi, in grado di accompagnare l'utente nelle fasi iniziali di interazione con il sistema.

Dal punto di vista dell'usabilità percepita, il sistema è stato generalmente valutato in modo positivo, soprattutto per quanto riguarda la semplicità delle azioni principali e la linearità del flusso di utilizzo. Tuttavia, la presenza di alcune incertezze iniziali indica l'importanza di ridurre il carico cognitivo richiesto nelle prime fasi di utilizzo.

### 3.4.3 Valutazione del sistema di crediti

Uno degli elementi più innovativi del progetto è rappresentato dal sistema di crediti, concepito come meccanismo di regolazione degli scambi tra utenti. Tale sistema si basa sull'idea che ogni contributo fornito (in termini di tutoraggio) generi un valore che può essere successivamente utilizzato per ottenere supporto.

I feedback raccolti mostrano che questo modello è stato generalmente compreso e percepito come equo, infatti, gli studenti hanno evidenziato come il sistema incentivi una partecipazione attiva e favorisca dinamiche di reciprocità. Allo stesso tempo, sono emerse alcune criticità legate alla piena comprensione del funzionamento operativo del sistema, in particolare per quanto riguarda le modalità di accumulo e utilizzo dei crediti. Questo aspetto evidenzia la necessità di rendere più trasparente e immediata la rappresentazione del sistema dei crediti all'interno dell'interfaccia.

### 3.4.4 Propensione alla partecipazione e dinamiche tra pari

Un ulteriore elemento di analisi riguarda la disponibilità degli studenti a partecipare attivamente al sistema, sia come utenti che richiedono supporto sia come tutor. I risultati indicano una maggiore propensione a chiedere aiuto piuttosto che a offrirlo, fenomeno coerente con dinamiche tipiche dei contesti educativi.

La disponibilità a svolgere il ruolo di tutor risulta fortemente influenzata dalla percezione delle proprie competenze. Gli studenti tendono infatti a proporsi come tutor principalmente in ambiti in cui si sentono sicuri, evidenziando l'importanza di un sistema che valorizzi e renda visibili tali competenze. Questi risultati suggeriscono che il successo del sistema dipende in larga misura dalla capacità di costruire un ambiente in cui gli utenti si sentano legittimati a condividere le proprie conoscenze, riducendo al contempo il timore di esposizione o giudizio.

### 3.4.5 Integrazione con lo spazio fisico di studio

Un aspetto distintivo del progetto GOAT è l'integrazione tra piattaforma digitale e spazio fisico di studio condiviso. I feedback raccolti indicano che questa componente è percepita come particolarmente rilevante, in quanto consente di superare alcune limitazioni tipiche dello studio individuale.

Gli studenti hanno evidenziato come la presenza di un ambiente dedicato allo studio possa favorire la concentrazione e aumentare la motivazione. Inoltre, la possibilità di interagire direttamente con altri studenti in un contesto strutturato è stata valutata positivamente, soprattutto in relazione alla dimensione sociale dell'apprendimento.

La presenza di figure adulte di supporto è stata anch'essa considerata un elemento importante, in quanto contribuisce a garantire un ambiente sicuro e organizzato, riducendo potenziali criticità legate alla gestione autonoma degli spazi.

### **3.4.6 Sintesi dei risultati e implicazioni progettuali**

Nel complesso, la fase di validazione ha evidenziato diversi punti di forza del progetto, tra cui:

- la coerenza del modello di scambio tra pari,
- la percezione di utilità dell'applicazione nel supporto allo studio,
- l'efficacia dell'integrazione tra componente digitale e fisica.

Parallelamente, sono emerse alcune aree di miglioramento:

- la necessità di semplificare la fase iniziale di utilizzo,
- il miglioramento della comunicazione del sistema di crediti,
- il rafforzamento dei meccanismi che incentivano il ruolo di tutor.

Questi risultati confermano la validità delle scelte progettuali descritte nel Capitolo 2, pur evidenziando la necessità di un'ulteriore iterazione del sistema, in linea con un approccio progettuale centrato sull'utente.

### **3.4.7 Considerazioni sui risultati**

Dall'analisi complessiva dei grafici e delle risposte emerge che:

- l'app è generalmente intuitiva;
- il sistema dei crediti è apprezzato ma migliorabile;
- il progetto è percepito come utile;
- l'utilizzo dipende molto dalla diffusione della piattaforma.

### 3.4.8 Conclusioni sul focus group in ambito scolastico

Il focus group condotto nelle scuole ha rappresentato un momento fondamentale nel processo di sviluppo del progetto GOAT, consentendo di confrontare il modello teorico con il contesto reale di utilizzo. I dati raccolti non solo validano l'impianto generale del sistema, ma forniscono anche indicazioni concrete per il suo miglioramento.

In questo senso, la validazione non si configura come una fase conclusiva, ma come parte integrante di un processo iterativo, in cui il feedback degli utenti assume un ruolo determinante nell'evoluzione del progetto.

## 3.5 Sicurezza e conformità al GDPR

Un sistema che raccoglie dati di studenti, e che in più gestisce un meccanismo di crediti virtuali, deve fare i conti anche con il GDPR, il Regolamento Generale sulla Protezione dei Dati (Regolamento UE 2016/679)[9]. Anche trattandosi di un prototipo realizzato per la tesi, ha senso fermarsi a guardare a che punto si trova GOAT rispetto a questo regolamento, dicendo onestamente cosa è già a posto e cosa invece manca.

I dati che GOAT raccoglie sono nome, email, password (salvata come hash), ruolo, materia principale, livello, crediti accumulati e lo storico delle attività svolte. Sono dati personali secondo l'art. 4 del GDPR, perché permettono di identificare l'utente. Dato che si tratta soprattutto di studenti minorenni o appena maggiorenni, bisognerebbe prestare particolare attenzione all'art. 8 del GDPR sul consenso dei minori, che in Italia fissa a 14 anni l'età da cui un ragazzo può dare da solo il consenso al trattamento dei propri dati nei servizi online.

Sul fronte dell'autenticazione, qualcosa di buono c'è già: le password non vengono mai salvate in chiaro, ma passano attraverso la funzione `password_hash()` di PHP, che usa l'algoritmo `bcrypt`, e vengono verificate al login con `password_verify()`. Questo è in linea con quanto richiesto dall'art. 32 del GDPR sulla sicurezza del trattamento, ed è una misura che protegge le credenziali anche se qualcuno riuscisse ad accedere al database senza autorizzazione.

Detto questo, ci sono diversi punti dove il prototipo non è ancora conforme al GDPR, ed è giusto dirlo chiaramente:

- **Le comunicazioni non sono cifrate:** il frontend e il backend si parlano tramite HTTP e non HTTPS, perché finora il prototipo è stato

sviluppato e testato solo in locale. Questo vuol dire che, se il sistema girasse su una rete non sicura, i dati scambiati (incluse le credenziali di login) potrebbero essere intercettati da terzi. Passare a HTTPS con un certificato TLS è il minimo richiesto dall'art. 32 del GDPR per qualsiasi servizio che tratta dati personali in produzione.

- **Manca un'informativa sulla privacy:** in fase di registrazione non viene spiegato all'utente quali dati vengono raccolti, perché, per quanto tempo vengono conservati e chi può vederli, come richiesto dagli artt. 13 e 14 del GDPR. Non c'è nemmeno un modo per raccogliere il consenso esplicito al trattamento dei dati.
- **Manca il diritto all'oblio:** l'art. 17 del GDPR dà all'utente il diritto di chiedere la cancellazione dei propri dati. Al momento GOAT non ha una funzione che permetta di eliminare il proprio account e tutto ciò che è collegato. L'unica forma di cancellazione presente nel sistema, quella che imposta `is_visible` a 0, riguarda gli annunci e non i dati personali dell'utente.
- **Manca una valutazione d'impatto:** un sistema vero che tratta dati di minori dovrebbe avere una valutazione d'impatto sulla protezione dei dati (DPIA, art. 35 del GDPR), cosa che non ha senso fare per un prototipo di tesi ma che andrebbe fatta prima di un eventuale utilizzo reale.
- **Le credenziali dell'admin sono scritte nel codice:** come già accennato, l'accesso alla dashboard amministrativa avviene con credenziali fisse inserite direttamente nel codice. Per un prototipo va bene, ma se il sistema fosse esposto al pubblico sarebbe un punto debole, perché non rispetta il principio di sicurezza fin dalla progettazione richiamato dall'art. 25 del GDPR.

In sintesi, GOAT ha già qualche accorgimento di sicurezza, come l'hashing delle password e l'uso di prepared statement per evitare l'SQL injection<sup>1</sup>, ma non si può ancora dire conforme al GDPR. Questi limiti non sono scelte definitive, ma derivano dal fatto che il progetto, essendo un prototipo di tesi, si è concentrato sulla validazione del modello concettuale e funzionale, lasciando da parte per ora gli aspetti normativi. Se in futuro GOAT dovesse diventare un sistema realmente utilizzato, andrebbero sicuramente aggiunti HTTPS,

---

<sup>1</sup>L'SQL injection (SQLi) è una vulnerabilità che permette a un utente malintenzionato di inserire comandi SQL dannosi all'interno dei campi di input di un'applicazione (come moduli di login o barre di ricerca). Questo manipola le query originali, consentendo di accedere, modificare o cancellare dati riservati senza autorizzazione.

un'informativa privacy, un modo per gestire il consenso e delle funzioni che permettano all'utente di esercitare i propri diritti previsti dal GDPR.



# Conclusioni

La realizzazione di questo progetto mi ha permesso di comprendere quanto l'informatica possa andare oltre il semplice sviluppo tecnico di un'applicazione. Durante il lavoro di progettazione e implementazione di GOAT è emerso infatti come, dietro ad ogni scelta tecnica, esista sempre anche una componente umana e sociale da tenere in considerazione. L'idea alla base della piattaforma non era soltanto quella di creare uno strumento digitale funzionante, ma soprattutto quella di provare a costruire uno spazio capace di favorire collaborazione, inclusione e partecipazione all'interno dell'ambiente scolastico. Nel contesto attuale molti studenti vivono la scuola in modo sempre più individuale e frammentato. Spesso mancano strumenti semplici che permettano di creare connessioni tra persone con interessi, necessità o difficoltà simili. Attraverso GOAT ho cercato quindi di immaginare una soluzione che potesse facilitare questi rapporti, sfruttando la tecnologia non come fine, ma come mezzo per migliorare la comunicazione e il supporto reciproco tra studenti. Tra gli aspetti che considero più importanti di questo lavoro riguarda proprio il potenziale impatto sociale che una piattaforma del genere potrebbe avere, anche funzionalità apparentemente semplici, come la creazione di gruppi studio o la possibilità di condividere attività ed eventi, possono contribuire a creare un ambiente più partecipativo e meno isolante, soprattutto per gli studenti che tendono ad avere maggiori difficoltà nell'integrarsi o nel trovare occasioni di confronto con gli altri. Un altro elemento fondamentale è stato il confronto con i feedback ricevuti durante la validazione del progetto. Vedere che gli studenti hanno compreso rapidamente il funzionamento della piattaforma e hanno mostrato interesse verso l'idea proposta mi ha fatto capire che esiste realmente la necessità di strumenti di questo tipo. Allo stesso tempo, i suggerimenti ricevuti hanno evidenziato come un progetto software non sia mai realmente concluso, ma rappresenti piuttosto un sistema in continua evoluzione che può essere migliorato e adattato nel tempo.

Come già specificato GOAT rimane ancora un progetto con ampi margini di crescita. Sarebbe interessante in futuro ampliare le funzionalità della piattaforma, migliorare ulteriormente l'esperienza utente e integrare strumen-

ti più avanzati per la gestione delle attività scolastiche e della collaborazione tra studenti. Tuttavia, anche nello stato attuale, il progetto dimostra come sia possibile utilizzare le tecnologie moderne per provare a rispondere a problematiche reali presenti nel mondo scolastico.

Dal punto di vista personale, questa esperienza ha rappresentato molto più di un semplice progetto di tesi. Mi ha permesso di mettere in pratica competenze tecniche legate allo sviluppo software, alla progettazione di database e all'organizzazione di un sistema completo, ma soprattutto mi ha insegnato quanto sia importante progettare pensando alle persone che utilizzeranno realmente ciò che viene creato. Ho capito che sviluppare un'applicazione non significa solamente scrivere codice, ma anche analizzare problemi concreti, comprendere i bisogni degli utenti e cercare di trovare soluzioni che siano realmente utili e accessibili.

In conclusione, questa tesi mi ha permesso non soltanto di consolidare competenze informatiche e progettuali, ma anche di riflettere sul ruolo che la tecnologia può avere nel migliorare le relazioni e l'esperienza quotidiana delle persone. GOAT rappresenta quindi non soltanto un'applicazione sviluppata a scopo didattico, ma un'idea che prova a mettere al centro collaborazione, condivisione e partecipazione, mostrando come anche un progetto nato in ambito universitario possa trasformarsi in qualcosa con un valore concreto e potenzialmente utile per la comunità.

# Bibliografia

- [1] Anselmo Penna. Dispersione scolastica e aumento della criminalità, c'è relazione. un problema europeo, 2025. URL: <https://www.orizzontescuola.it/dispersione-scolastica-e-aumento-della-criminalita-ce-relazione-un-problema->
- [2] Davide Falco. Appunti di sviluppo applicazioni software. URL: <https://davidefalco.github.io/appunti-informatica/sviluppo-applicazioni-software/00-processi-per-lo-sviluppo-software/sviluppo-incrementale.html>.
- [3] Ionic Framework. Ionic angular overview. URL: <https://ionicframework.com/docs/angular/overview>.
- [4] Lucid Charts. Cos'è l'unified modeling language. URL: <https://www.lucidchart.com/pages/it/uml>.
- [5] Marta Chemello, Cesare Cornoldi. Come utilizzare il peer tutoring in classe con alunni con bisogni educativi speciali. URL: <https://www.erickson.it/it/mondo-erickson/come-utilizzare-il-peer-tutoring-in-classe-con-alunni-con-bisogni-educativi->
- [6] Martina Giunchi. Progetto GOAT, 2025.
- [7] Mia-Platform Team. Cos'è l'architettura composable? URL: <https://mia-platform.eu/it/blog/architettura-composable/>.
- [8] OpenPolis. Come la condizione familiare incide sugli apprendimenti degli studenti, 2022. URL: <https://www.openpolis.it/come-la-condizione-familiare-incide-sugli-apprendimenti-degli-studenti/>.
- [9] Parlamento Europeo e Consiglio dell'Unione Europea. Regolamento (ue) 2016/679 del parlamento europeo e del consiglio del 27 aprile 2016 relativo alla protezione delle persone fisiche con riguardo al trattamento dei dati

- personali (gdpr). URL: <https://eur-lex.europa.eu/legal-content/IT/TXT/?uri=CELEX:32016R0679>.
- [10] Regione Emilia-Romagna. Giovani NEET, 2024. URL: <https://statistica.regione.emilia-romagna.it/dati/fb/lavoro/neet>.
- [11] TGCOM24. Tra i giovani spopolano le app di dating: ma solo per amicizia, 2025. URL: [https://www.tgcom24.mediaset.it/tgtech/app-giovani-dating-amici-timeleft-bumble\\_99405803-202502k.shtml](https://www.tgcom24.mediaset.it/tgtech/app-giovani-dating-amici-timeleft-bumble_99405803-202502k.shtml).
- [12] Tomas Cipriani. Peer education e peer tutoring. URL: <https://tomascipriani.it/peereducation/>.
- [13] Visur Resolutions. Requisiti funzionali e non funzionali. URL: <https://visuresolutions.com/it/alm-guide/functional-vs-non-functional-requirements/>.