



# Progetto di Ingegneria del Software 2025/26

**Università Ca' Foscari Venezia**

**Documento di Progettazione**

**EcoGroup**

**30/11/2025**



### Document Informations

Nome Progetto	Acronimo
Deliverable	Documento di Progettazione
Data di Consegnna	30/11/2025
Team Leader	Enrico Sforza
Team members	Enrico Rampazzo Alberto Ferragosti

### Document History

Version	Issue Date	Stage	Changes	Contributors
1.0	28/11/2025	Draft	Scritta sezione 2, 3, 4, 5 parte Profilo	902600
1.1	30/11/2025	Draft	Completate sezioni 2,3,4,5 parte registrazione utente e login	895936
1.2	30/11/2025	Draft	Completate le sezioni 3,4,5 parte Quest	901141
2.0	25/01/2026	final	Revisione pre consegna	895936



# Indice

<b>1. Introduzione</b>	<b>4</b>
<b>2. Architettura del sistema</b>	<b>4</b>
2.1 Componenti e Responsabilità	4
Principi di progettazione adottati	4
Coesione	4
Coupling (Accoppiamento)	5
Modello architettonico orientato agli oggetti	5
2.2 Diagramma di Distribuzione	5
<b>3. Modello di controllo e dei dati</b>	<b>6</b>
3.1 Modello dei Dati	6
Modello di controllo adottato	6
1. Controllo event-driven (lato client)	6
2. Controllo call-return (client → server)	7
3. Livello di coordinamento	7
Profilo	7
Registrazione / Creazione Utente	8
Login	9
Quest	9
3.2 Modello del Controllo	11
Registrazione / Creazione Utente	12
Login	14
Quest	15
<b>4. Modelli UML</b>	<b>17</b>
Profilo	17
Registrazione / Creazione Utente – Diagramma di Sequenza	18
Login – Diagramma di Sequenza	18
Attributi di qualità della progettazione	19
Quest	19
<b>5. Interfaccia utente</b>	<b>21</b>
Profilo	21
Registrazione / Creazione Utente	21
Login	22
Quest	23



## 1. Introduzione

Il sistema progettato è un'applicazione mobile Android finalizzata a promuovere comportamenti ecosostenibili attraverso meccaniche di **gamification**.

L'applicazione permette agli utenti di:

- **Svolgere Missioni (Quest):** Selezionare e completare obiettivi ecologici giornalieri o ricorrenti (es. mobilità sostenibile, riduzione rifiuti).
- **Monitorare i Progressi:** Visualizzare tramite il proprio profilo il punteggio accumulato, il livello raggiunto e la stima della CO<sub>2</sub> risparmiata grazie alle attività svolte.
- **Interagire con la Community:** Confrontare i propri risultati con quelli della lista amici per incentivare la partecipazione attiva.

## 2. Architettura del sistema

Il sistema EcoApp adotta un'architettura **Client-Server** a due livelli. Questa scelta progettuale è motivata dalla necessità di centralizzare la logica di business e i dati condivisi, come classifiche e profili utente, pur mantenendo sul dispositivo mobile la gestione dell'interattività e l'accesso diretto ai sensori.

### 2.1 Componenti e Responsabilità

Il **Client**, costituito dall'applicazione Android, svolge il ruolo di front-end e si occupa dell'acquisizione dei dati. Questa componente sfrutta l'hardware del dispositivo, in particolare GPS e accelerometro, per rilevare le attività ecologiche e offrire un'interfaccia reattiva all'utente. Tra le sue funzioni chiave rientrano la gestione dell'interfaccia utente (UI), la lettura dei sensori e la gestione di una cache locale, fondamentale per garantire l'usabilità offline come richiesto dal requisito RNF7.

Il **Server**, composto da Backend e DBMS, si occupa della validazione e della persistenza dei dati. La scelta di separare questo livello è dettata dall'esigenza di garantire la sicurezza, impedendo la manipolazione dei punteggi lato client, e l'integrità dei dati, fungendo da unica fonte di verità (*single source of truth*) per gli account e le quest. Il server gestisce la comunicazione, esegue la logica di calcolo dei punti e amministra il database centralizzato.

### Principi di progettazione adottati

La progettazione del sistema segue i principi illustrati nel modulo teorico, con particolare attenzione a:

#### Coesione

Ogni componente è progettato affinché svolga un insieme di compiti strettamente correlati.

- La **RegistrationView**, la **LoginView** e la **ProfileView** gestiscono esclusivamente la presentazione dei dati.



- I controller (**AuthController**, **ProfileController**) encapsulano interamente la logica applicativa relativa al proprio modulo.
- Le classi di dominio (**Utente**, **StatisticheUtente**, **Sessione**) mantengono uno stato coerente e ben definito.

Questo approccio garantisce l'elevata coesione raccomandata nelle slide del docente, facilitando estensioni e manutenzione del sistema.

## Coupling (Accoppiamento)

Il sistema minimizza l'accoppiamento tra componenti utilizzando interfacce chiare e comunicazioni standardizzate:

- il client comunica con il backend *solo* tramite REST API;
- la cache locale è isolata e non dipende dall'implementazione del server;
- le classi Utente / StatisticheUtente / Sessione non sono dipendenti dai dettagli della persistenza.

Questa separazione riduce propagazioni di errore e favorisce modifiche localizzate, in linea con gli attributi di qualità descritti nelle slide.

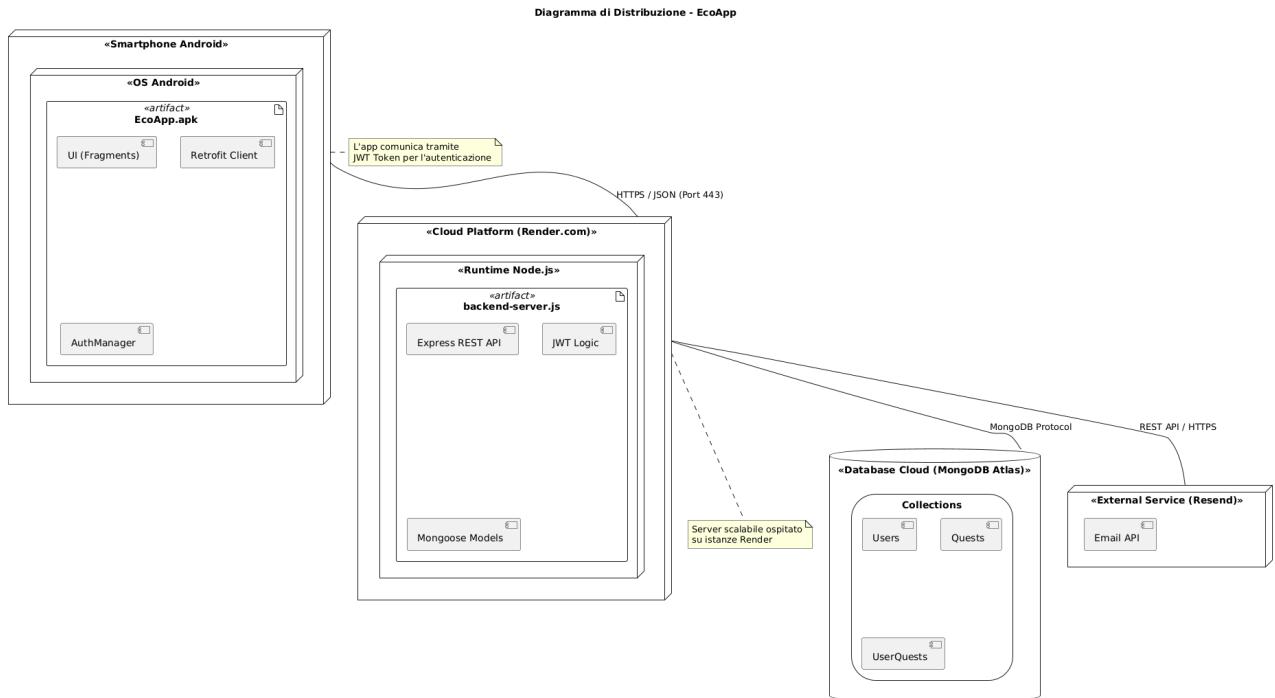
## Modello architetturale orientato agli oggetti

Il sistema adotta un approccio **object-oriented decentralizzato**, dove ogni entità (Utente, Sessione, QuestCompletata...) mantiene il proprio stato e offre i metodi per manipolarlo.

Questo è coerente con quanto indicato nelle slide, che distinguono l'OO come modello in cui il controllo non è centralizzato ma distribuito tra gli oggetti che cooperano attraverso messaggi/metodi.



## 2.2 Diagramma di Distribuzione



## 3. Modello di controllo e dei dati

### 3.1 Modello dei Dati

#### Modello di controllo adottato

Il sistema utilizza due modelli di controllo complementari, come presentato nel materiale teorico:

##### 1. Controllo event-driven (lato client)

Le interazioni nell'applicazione Android seguono un modello **event-driven**, nel quale:

- ogni azione dell'utente (tap, inserimento testo, conferma) genera un **evento**;
- questo attiva il relativo controller (es. AuthController, ProfileController);
- il controller coordina la logica applicativa ed eventuali chiamate al server.

Questo approccio è quello indicato nelle slide come tipico dei sistemi decentralizzati e interattivi.

##### 2. Controllo call-return (client → server)



La comunicazione tra l'app e il backend avviene con lo schema **call-return** tipico delle REST API:

- il client invia una richiesta (POST /auth/login, POST /auth/register, GET /profile),
- il server elabora e restituisce una risposta strutturata.

Questo modello è coerente con il controllo “call-return” presentato nelle slide (push del controllo in profondità e ritorno gerarchico della chiamata).

### 3. Livello di coordinamento

Il sistema non utilizza modelli broadcast o interrupt-driven:

- non c’è un unico controllore centralizzato;
- non ci sono notifiche asincrone che scatenano automaticamente il controllo.

Il controllo è invece **decentralizzato**, coerente con il modello OO

#### Profilo

Il modello dei dati per la sezione **Profilo** è stato progettato con un’architettura modulare e disaccoppiata, ponendo la classe **Utente** al centro come aggregatore primario delle informazioni essenziali. Questa struttura garantisce non solo la coerenza dei dati anagrafici e di accesso, ma anche la flessibilità necessaria per l’evoluzione futura della piattaforma e la personalizzazione avanzata dell’esperienza utente. Il disaccoppiamento è cruciale: mentre l’entità **Utente** gestisce l’identità e la personalizzazione diretta, aspetti dinamici e in continua evoluzione, come la **gamification** e l’**impatto ambientale**, sono delegati all’entità sussidiaria **StatisticheUtente**. Inoltre, la tracciabilità e la trasparenza delle azioni passate sono affidate alla classe **QuestCompletata**, che funge da registro storico.**Entità Principali e Dettagli del Modello**

Le tre entità fondamentali identificate per il frontend, in particolare per la visualizzazione all’interno del **ProfileFragment**, sono dettagliate come segue:**1. Utente (Identità e Personalizzazione)**

Questa classe rappresenta l’identità digitale dell’account e funge da ponte tra i dati di autenticazione del backend e gli elementi visivi dell’intestazione del profilo.

- **Identità e Riconoscimento:**

- **name** (Nome Completo): Il campo formale, spesso utilizzato in comunicazioni ufficiali o intestazioni principali.
- **nickname** (Identificatore Sociale Unico): L’handle o pseudonimo dell’utente, cruciale per l’interazione sociale all’interno della piattaforma (tagging, ricerca amici). La sua unicità è garantita a livello di sistema.



- **Personalizzazione del Profilo:**
  - **bio** (Bio Testuale): Un campo di testo libero (con un limite di caratteri definito) che permette all'utente di esprimere la propria missione o il proprio interesse ecologico.
  - **urlImmagineProfilo**: Un URL che punta all'asset grafico (avatar) dell'utente. La gestione del caricamento e dell'ottimizzazione dell'immagine è gestita dal frontend, ma il riferimento persistente è mappato qui.
- **Social Metrics di Base:**
  - **followingCount**: Un contatore che indica il numero di altri utenti seguiti. Questo dato è un'aggregazione derivata dalla lista completa degli "amici" o "follower" dell'utente e mira a visualizzare immediatamente l'ampiezza e l'attività della rete sociale dell'utente.

## 2. Statistiche Utente (Progresso e Impatto)

Questa classe incapsula tutte le metriche relative al progresso dell'utente, ai traguardi raggiunti e all'impatto ecologico generato. È cruciale notare che i dati contenuti in questa entità sono estratti dal server in modalità **sola lettura** da parte del frontend, garantendo la massima coerenza e integrità del sistema di premi e gamification.

- **Sistemi di Gamification:**
  - **level** (Livello Attuale): L'attuale grado di esperienza dell'utente nella piattaforma, spesso legato a soglie di punteggio.
  - **totalPoints** (Punteggio Totale): La somma cumulativa di tutti i punti guadagnati attraverso il completamento delle attività. Questi due attributi sono la fonte primaria per la popolamento delle **barre di avanzamento** e della logica di *level-up*.
- **Feedback sull'Impatto Ecologico:**
  - **co2Saved** (CO<sub>2</sub> Risparmiata): Una metrica fondamentale, rappresentata come valore numerico (es. in chilogrammi o tonnellate), che fornisce un feedback tangibile e quantificabile sul contributo dell'utente al risparmio di anidride carbonica.
- **Riconoscimenti e Traguardi (Achievements):**
  - **badges** (Array di Oggetti Riconoscimento): Una lista dinamica contenente oggetti che rappresentano i riconoscimenti o i traguardi sbloccati (es. "Eco-Novizio", "Maestro del Riciclo"). Ogni oggetto badge è composto da:
    - **nome**: La denominazione del riconoscimento.
    - **descrizione**: Il criterio di sblocco e una breve spiegazione del significato del traguardo.

## 3. QuestCompletata (Storico delle Attività)

Questa entità agisce come un registro cronologico che permette all'utente di esaminare la propria storia di interazione con la piattaforma e le missioni completate.

- **Tracciamento e Riferimento:**
  - **questId**: L'identificatore univoco della missione completata. Questo campo stabilisce il collegamento fondamentale tra l'istanza di completamento e la definizione (metadati) della missione stessa.



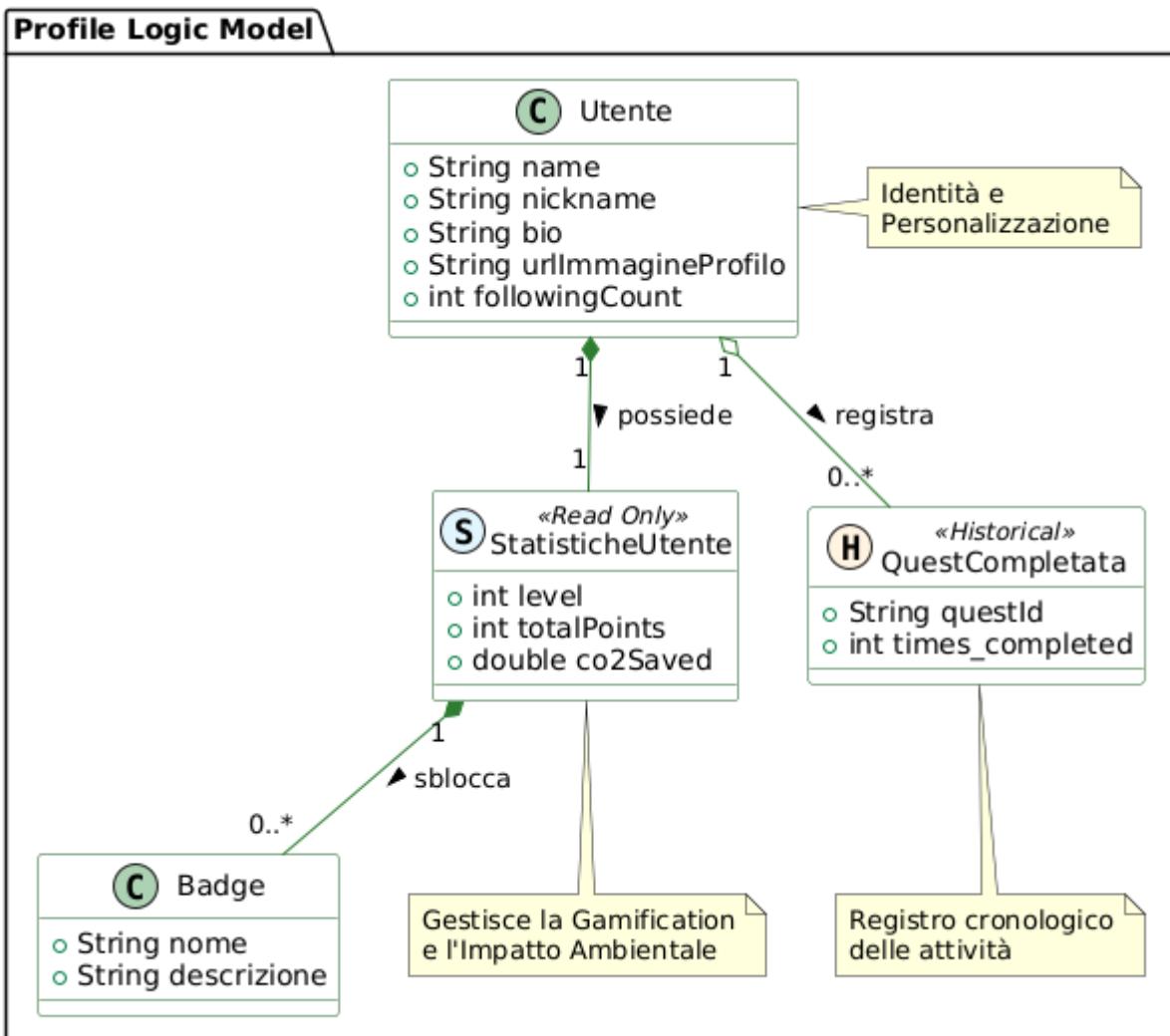
- **Metriche di Istanza e Ripetizione:**

- **times\_completed:** Un contatore che indica la frequenza con cui quella specifica missione (**questId**) è stata portata a termine dall'utente. Questo è particolarmente utile per le missioni ripetibili e per calcolare l'impegno costante.

- **Integrazione con la UI:**

- I dati di questa classe sono l'origine per la popolare la sezione "**Attività Recenti**" del frammento, fornendo un *feed* cronologico che non solo evidenzia l'impegno ecologico dell'utente, ma serve anche da motivazione e promemoria delle azioni intraprese.

In sintesi, il modello è stato concepito per separare chiaramente le preoccupazioni: **Identità (Utente)**, **Progresso e Premi (StatisticheUtente)** e **Storico Azioni (QuestCompletata)**, massimizzando l'efficienza della gestione dei dati e l'esperienza utente.





## Registrazione / Creazione Utente

Il modello dei dati per la registrazione è semplificato rispetto a pattern tradizionali con verifica email. La scelta architettonale privilegia l'immediatezza dell'onboarding.

**Utente** (classe centrale, riutilizzata dal modulo Profilo):

- email (string, unique, lowercase)
- password (string, hash bcrypt)
- name (string)
- nickname (string, opzionale)
- totalPoints (number, default 0)
- - co2Saved (number, default 0)
- level (string, default "Eco-Novizio")
- badges (array di Badge)
- friends (array di ObjectId)
- - createdAt, updatedAt (timestamps automatici Mongoose)

**Nota architetturale:**

- NON esiste uno stato account (attivo/in attesa/sospeso)
- L'account è IMMEDIATAMENTE ATTIVO dopo la registrazione
- L'email di benvenuto è opzionale e non bloccante (fire-and-forget via Resend API)
- NON esiste TokenVerificaEmail - nessuna verifica richiesta

Questa scelta semplifica l'onboarding riducendo l'attrito per l'utente, accettando il trade-off di non validare la proprietà dell'indirizzo email.

## Login

Il login utilizza un'architettura stateless basata su JWT (JSON Web Token).

**Autenticazione:**

- Il server genera un JWT firmato contenente: userId, email, iat, exp
- Durata token: 7 giorni (default) o 30 giorni (con checkbox "Ricordami")
- Il token è l'unico elemento di sessione - nessuna tabella SessioniAttive

**Persistenza lato client (TokenManager.java):**

- Token salvato in SharedPreferences (MODE\_PRIVATE)
- All'avvio dell'app, se presente token non scaduto → auto-login
- Se token scaduto → redirect a LoginActivity

Nota architetturale:

Ingegneria del Software 2025/26 – docente: prof. Agostino Cortesi

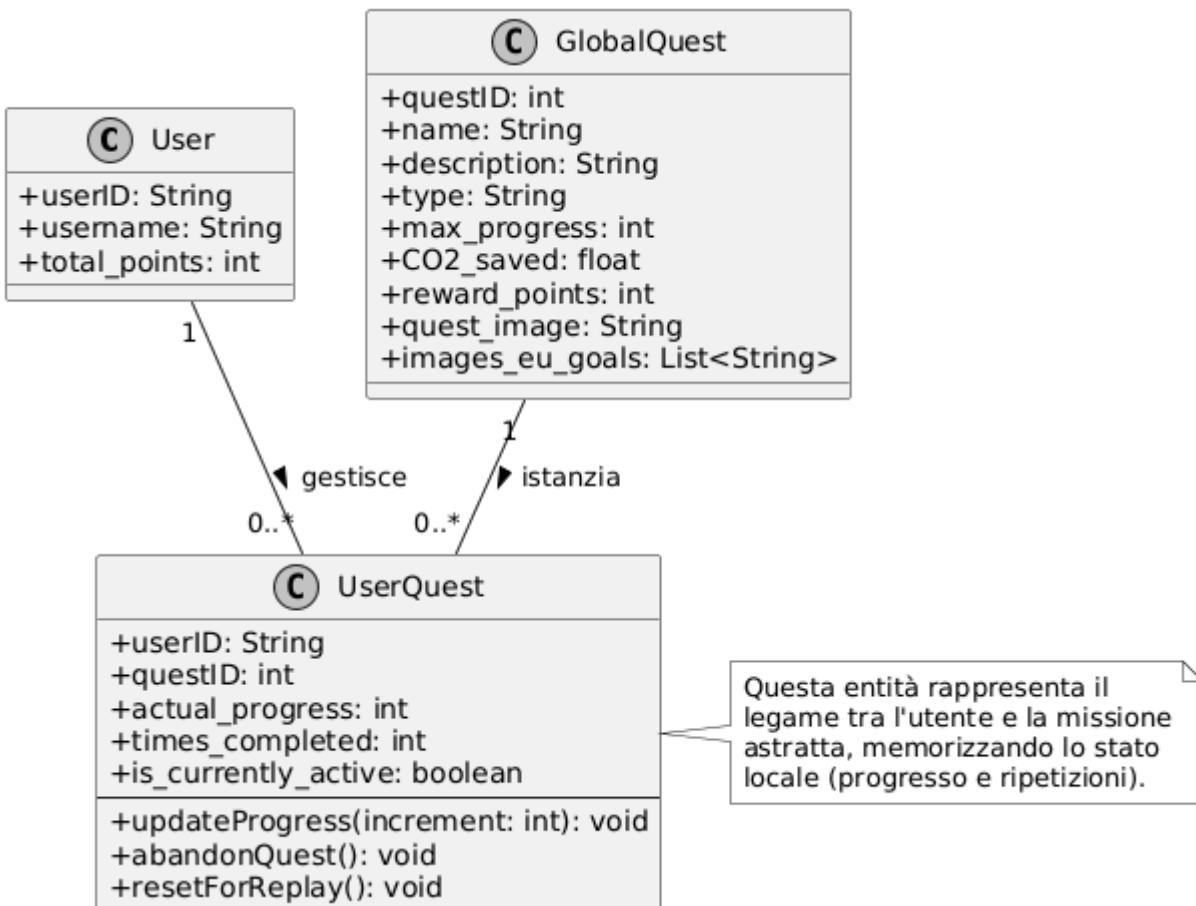


- NON esiste "silent refresh" - alla scadenza l'utente deve ri-autenticarsi
- NON esiste "logout da tutti i dispositivi" - JWT è stateless
- Il logout locale consiste nel cancellare il token da SharedPreferences

## Quest

Il modulo delle Quest introduce 2 entità principali, distinte per garantire coesione e separazione tra le quest globali uguali per tutti gli utenti e le quest locali formate dall'unione delle 2 seguenti entità.

- **GlobalQuest:** Rappresenta la missione “astratta”, definita dal sistema.
  - questID
  - name
  - description
  - type (es. mobilità, rifiuti, energia)
  - max\_progress
  - CO2\_saved
  - reward\_points
  - quest\_image
  - images Eu\_goals
- **UserQuest :** È l'assegnazione di una Quest a un utente. La sua funzione è modellare lo stato della missione.
  - userID
  - questID
  - actual\_progress
  - times\_completed
  - is\_currently\_active



## 3.2 Modello del Controllo

### Profilo

Il controllo della sezione "Profilo" (amici e richieste di amicizia) è implementato attraverso un sofisticato approccio di architettura ibrida, che coniuga i principi dell'interazione **event-driven** per la reattività dell'interfaccia utente (UI) con un modello **call-return asincrono** per la gestione delle operazioni di I/O (Input/Output) e la sincronizzazione dei dati con il backend.<sup>1</sup>. Visualizzazione e Aggiornamento dei Dati (Controllo Asincrono)

La fase iniziale di caricamento dei dati e i successivi aggiornamenti che richiedono comunicazione con il server si basano su un meccanismo di controllo asincrono per evitare il blocco del thread principale (UI thread).

- **Inizializzazione e Caricamento:** All'apertura della sezione "Profilo" (tipicamente gestita da un Fragment come FriendsFragment nel contesto Android), il controller avvia immediatamente due richieste asincrone parallele attraverso il modulo AuthService: loadFriends() e loadRequests().



Queste chiamate sono non bloccanti, consentendo alla UI di rimanere responsiva (ad esempio, mostrando un indicatore di caricamento).

- **Gestione della Risposta tramite Callback:** Il controllo gestisce il risultato delle richieste tramite un meccanismo di *callback*. Solo al ricevimento di una risposta positiva dal server (verificata tramite il flag `isSuccessful`), i dati deserializzati vengono elaborati.
- **Mappatura e Coerenza Locale:** I dati ricevuti vengono mappati in strutture dati locali persistenti (cache), come ad esempio `originalFriendsList`. Mantenere una copia locale (cache) è fondamentale per le operazioni successive di filtro e per garantire che la UI rifletta sempre l'ultimo stato noto.
- **Aggiornamento Reale della UI:** La modifica della lista locale innesca l'aggiornamento della vista tramite la notifica al relativo *Adapter* (ad esempio, `notifyDataSetChanged()`). Questo meccanismo garantisce che la visualizzazione venga aggiornata solo con dati validi e confermati.

## 2. Interazione e Ricerca (Controllo Event-Driven)

Le interazioni dell'utente che non richiedono l'immediata comunicazione con il server sono gestite con un approccio **event-driven**, mirato a massimizzare la velocità di risposta e l'esperienza utente.

- **Filtro e Ricerca Reattiva:** L'inserimento di testo nel campo di ricerca (Search Bar) agisce come un *evento*. Questo evento è intercettato da un `TextWatcher` dedicato, che attiva il metodo `filterFriends()`.
  - **Operazione su Cache Locale:** Cruciale è che l'operazione di filtro **non genera chiamate al server**. Il controllo opera esclusivamente sulla cache locale (`originalFriendsList`). Questo garantisce una risposta **istantanea** (near-zero latency) dell'interfaccia utente, un elemento chiave per l'usabilità.
  - **Dati Filtrati:** Il risultato del filtro viene presentato all'utente manipolando la lista visualizzata, senza alterare la lista originale non filtrata.
- **Navigazione Dettaglio:** Il tocco (tap) su un elemento profilo è un altro *evento*. Questo evento è delegato al `NavController` (nell'architettura basata su Android Jetpack Navigation). Il controllo in questo caso si limita a:
  - **Cambio di Stato:** Eseguire la transizione verso una nuova destinazione (es. `ProfileDetailFragment`).
  - **Propagazione del Contesto:** Propagare l'identificativo univoco dell'utente selezionato (es. `userId`) all'interno di un `Bundle`, fornendo il contesto necessario alla nuova destinazione per caricare i dati specifici.

## 3. Operazioni Transazionali e Modifiche di Stato (Pattern Command)

Le operazioni che modificano lo stato permanente dell'applicazione (es. rimozione di un amico, accettazione/rifiuto di una richiesta) sono incapsulate in un controllo rigoroso che segue i principi del **Pattern Command** per garantire integrità e coerenza.

- **Conferma Esplicita (Pre-Condizione):** Prima di inviare qualsiasi comando transazionale al backend (es. `removeFriend()`, `respondToRequest()`), il controllo interviene obbligatoriamente visualizzando un `AlertDialog` o un elemento di conferma. Questo serve a validare l'intenzione dell'utente e prevenire modifiche accidentali.



- **Sincronizzazione e Commit Condizionato:** L'aggiornamento dello stato locale (cache e UI) è subordinato al successo dell'operazione sul backend.
  - **Invio e Attesa:** Il comando viene inviato al backend. Il controllo rimane in uno stato di attesa (eventualmente mostrando uno stato "in sospeso").
  - **Aggiornamento Post-Successo:** Solo dopo aver ricevuto la conferma di successo (*success acknowledgment*) dal server, il controllo esegue la modifica dello stato locale e aggiorna la visualizzazione. Questo processo bidirezionale (backend -> client) garantisce la **coerenza forte** tra la cache del client e il database centrale.
- **Gestione degli Errori (Rollback e Feedback):** L'integrità del sistema è protetta da un meccanismo robusto di gestione degli errori.
  - **Intercettazione dell'Evento di Errore:** In caso di fallimento della richiesta o di problemi di connettività di rete (es. timeout), il metodo `handleErrorResponse()` intercetta l'evento.
  - **Feedback Visivo:** L'utente riceve immediatamente un feedback non intrusivo (es. un `Toast` con il messaggio d'errore).
  - **Annullamento della Modifica Locale:** Se un'operazione ha tentato una modifica preliminare sulla UI prima della conferma, questa viene annullata (rollback), ripristinando lo stato precedente e garantendo che la UI non mostri dati incoerenti o non confermati.

## Registrazione / Creazione Utente

Il controllo per la registrazione segue lo stesso approccio *event-driven* già usato per il Profilo: il flusso è guidato dagli eventi generati dalla UI (tap sui campi, pulsante “Registrati”, ecc.), mentre la logica applicativa vera e propria è centralizzata in un **AuthController** sul client, che dialoga con il backend tramite API REST sicure (HTTPS).

### Scenario Principale

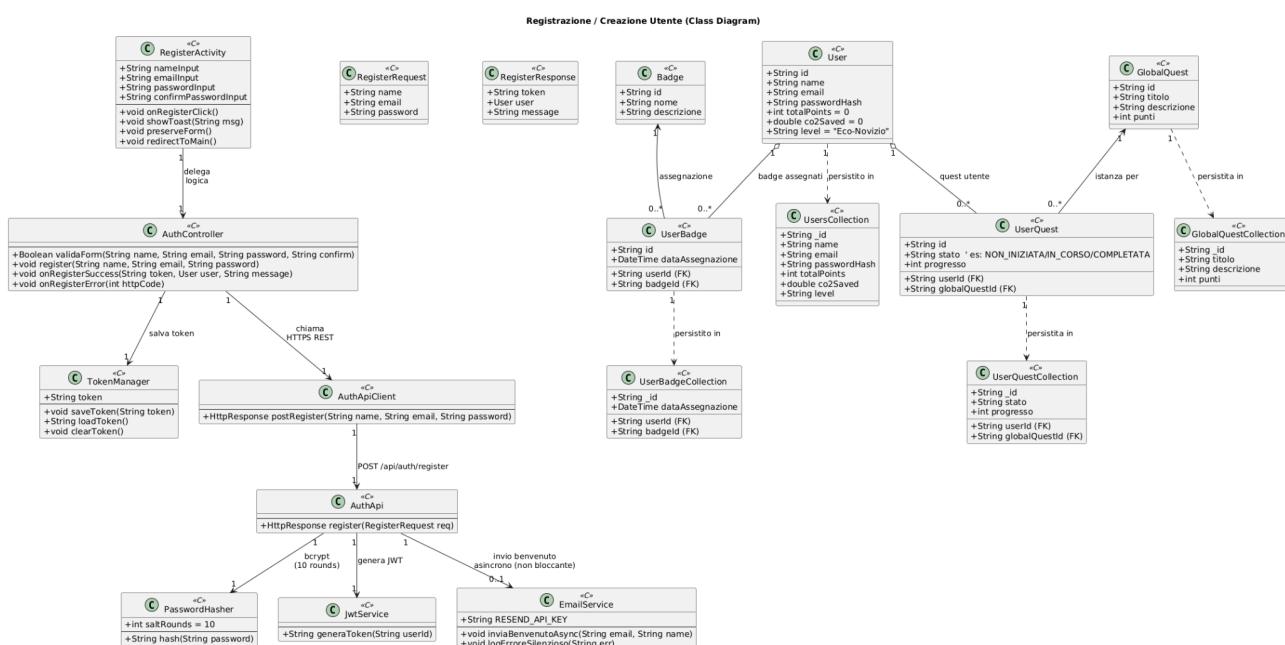
- Compilazione form
  - L'utente inserisce: nome, email, password, conferma password
- Validazione client-side:
  - Nome: obbligatorio, min 2 caratteri
  - Email: formato valido (Patterns.EMAIL\_ADDRESS)
  - Password: min 6 caratteri
  - Conferma: deve coincidere con password
- Invio richiesta
  - POST `/api/auth/register` con body: {name, email, password}
  - Nessun invio di conferma password (validata solo client-side)
- Validazione server
  - Verifica unicità email (case-insensitive)
  - Se email duplicata: HTTP 409 Conflict
- Creazione account (immediata)
  - Password hashata con bcrypt (10 salt rounds)
  - Documento User creato in MongoDB con valori default:
  - totalPoints: 0, co2Saved: 0, level: "Eco-Novizio"
  - Badge iniziale assegnato



- UserQuest inizializzate per tutte le GlobalQuest esistenti
- JWT generato immediatamente
- Email di benvenuto (asincrona, non bloccante)
  - RESEND\_API\_KEY configurata: invio email benvenuto
  - Fallimento email NON blocca la registrazione
  - Errori loggati silenziosamente
- Risposta e navigazione
  - HTTP 201 con: {token, user, message}
  - Client salva token in Shared Preferences
  - Redirect a MainActivity (login automatico post-registrazione)

#### Gestione errori:

- Email duplicata → Toast "Email già registrata", form preservato
- Errore rete → Toast "Errore di connessione", form preservato
- Errore server → Toast con messaggio generico



#### Login

Anche il flusso di login è guidato da eventi della UI ma con una logica leggermente diversa, perché deve gestire sia il primo accesso sia la riapertura di sessioni salvate.

#### Scenario Principale

- Inserimento credenziali
  - Email e password nella LoginActivity
  - Checkbox "Ricordami" (opzionale)
- Validazione client



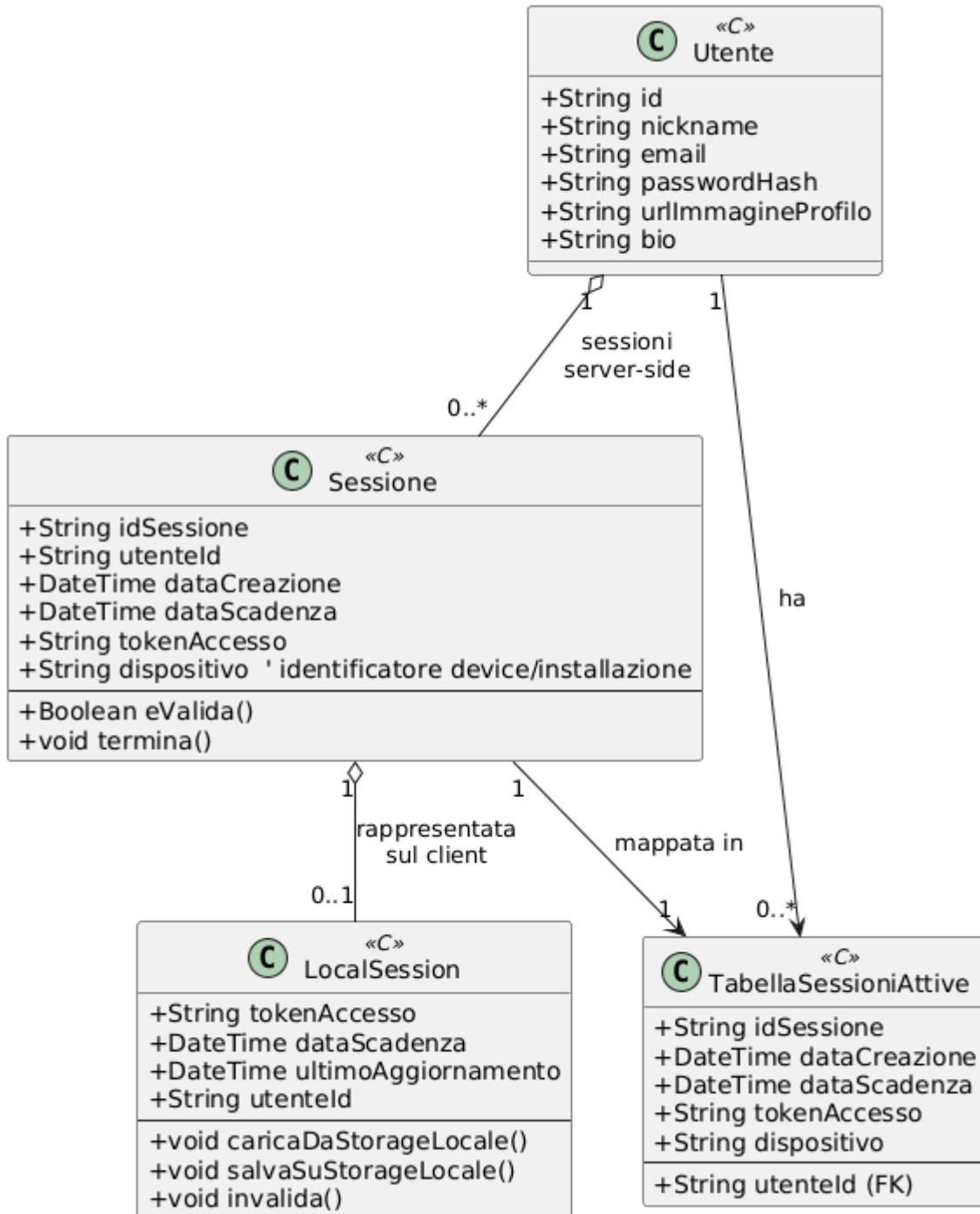
- Campi non vuoti
- Formato email valido
- Richiesta autenticazione
  - POST `/api/auth/login` con: {email, password, rememberMe}
- Verifica server
  - lookup utente per email (case-insensitive)
  - Confronto password con hash bcrypt
  - Se credenziali errate: HTTP 401
- Rate limiting
  - Dopo 5 tentativi falliti: blocco 15 minuti
  - Implementato con express-rate-limit
- Generazione sessione
  - JWT creato con durata:
  - 7 giorni (default)
  - 30 giorni (se rememberMe=true)
- Risposta e navigazione
  - HTTP 200 con: {token, user}
  - Token salvato in SharedPreferences
  - Redirect a MainActivity

#### Gestione errori:

- Credenziali errate → "Email o password non corretti"
- Rate limit → "Troppi tentativi. Riprova tra 15 minuti"
- Errore rete → "Errore di connessione"
- Auto-login all'avvio:
  - TokenManager verifica presenza token in SharedPreferences
  - Se presente e non scaduto → skip LoginActivity, vai a MainActivity
  - Se assente o scaduto → mostra LoginActivity



## Modello Login / Sessioni (Class Diagram)



### Quest

Il controllo delle quest segue il modello ibrido già visto per Profilo/Registrazione.



1. **Controllo event-driven (lato client):** Ogni azione nella UI genera un evento gestito dal QuestController.
  - 1.1. Possibili eventi generati dall'utente:
    - 1.1.1. Tap su "Accetta quest".
    - 1.1.2. Conferma manuale dell'utente (es. foto, check).
  - 1.2. QuestController:
    - 1.2.1. Chiama il server per confermare l'azione quando online.
2. **Controllo call-return (client → server):** Il backend ha la responsabilità finale della validazione.  
Flusso tipo:
  - 2.1. @GET("api/quests") : Il server restituisce le quest globali.
  - 2.2. @GET("api/user/quests"): Il server restituisce le quest locali.
  - 2.3. @POST("api/user/quests/update"): Avviene una scrittura nel server nel momento in cui una quest diventa "ongoing".
  - 2.4. @POST("api/user/quests/set-first-activation"): Per scrivere nel server nel momento in cui si forma il primo collegamento tra una quest globale e un utente.
  - 2.5. @POST("api/user/quests/set-quest-parameters"): Per modificare i progressi di un utente.
3. **Gestione offline:** Coerente con RNF7 e con il comportamento del Profilo.
  - 3.1. Le quest *accettate* sono in cache.
  - 3.2. Le quest *in attesa di conferma* vengono marcate come "pending".
  - 3.3. Alla riconnessione: il QuestController invia tutte le completion pending → il server valida → aggiorna cache.
4. **Gestione periodicità (ricorrenti):** Il server gestisce completamente la logica di rinnovo per evitare manipolazioni locali.
  - 4.1. Una volta completata una quest è subito possibile ripeterla.
5. **Gestione errori**
  - 5.1. Condizioni non soddisfatte: Errore "dati insufficienti".
  - 5.2. Quest scaduta: il server la marca come "scaduta".
  - 5.3. Tentativo di completamento duplicato: risposta "già completata".

#### Scenario completo (flusso principale):

- **Assegnazione:** L'utente apre la sezione delle quest globali → il client invia GET → il server restituisce le QuestIstanziate.
- **Accettazione:** UI genera evento "Accetta" → QuestController → POST al backend → DB registra l'istanza.
- **Rilevamento attività:** Eventi generati dai sensori o dalla UI → QuestController aggiorna progressi locali.
- **Completamento:**
  - QuestController invia POST /completa con i dati.

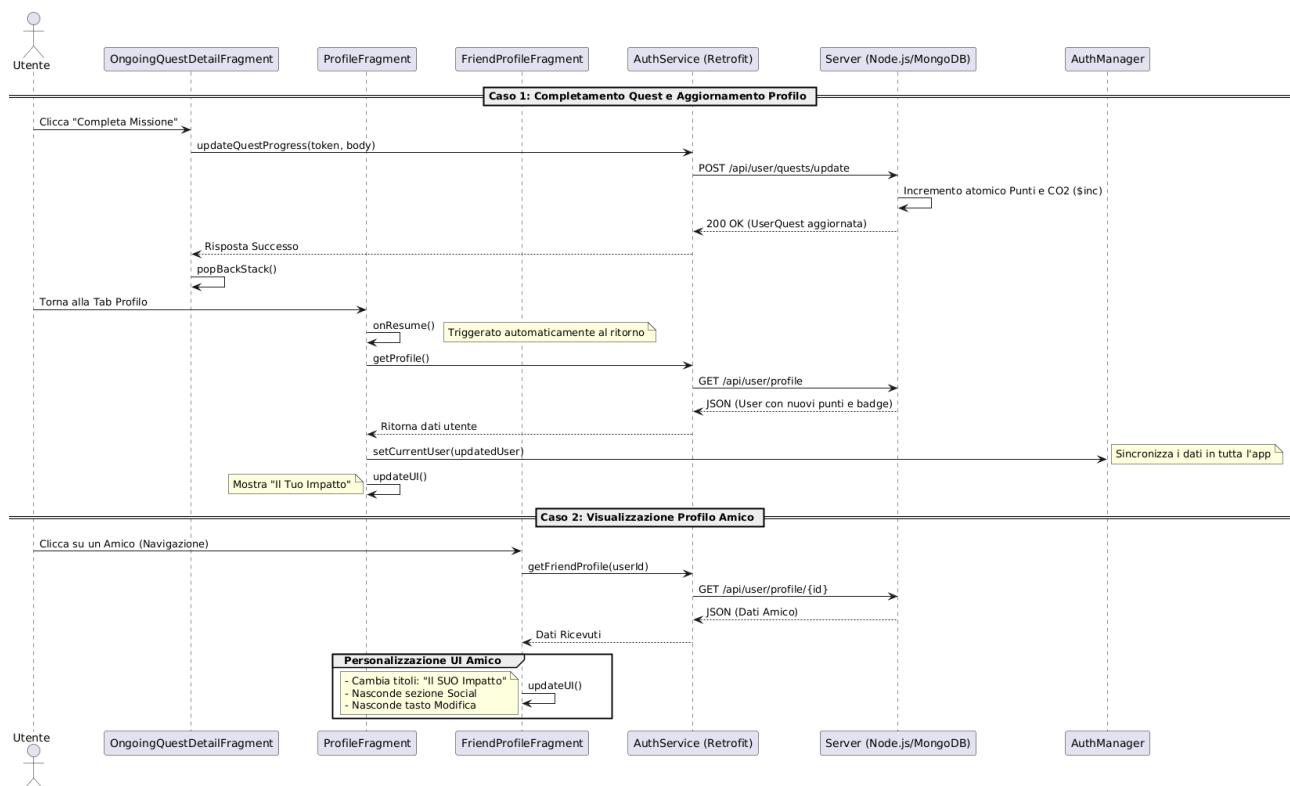


- Server valida → registra QuestCompletata → aggiorna StatisticheUtente → risposta con punti.
- **Aggiornamento cache:** Il client aggiorna la visualizzazione e lo storico locale.

## 4. Modelli UML

### Profilo

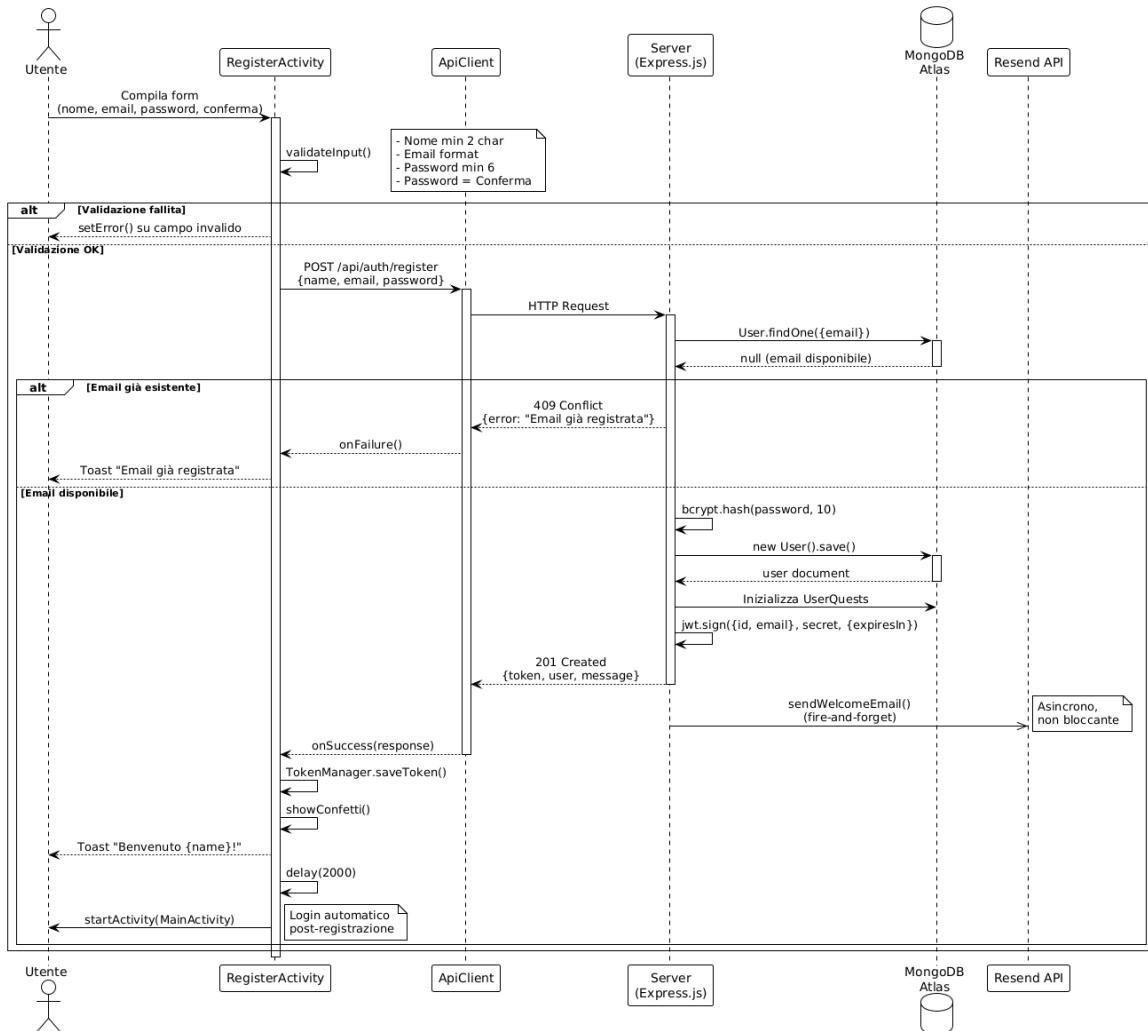
Il diagramma seguente descrive il flusso di operazioni per la visualizzazione del profilo utente



### Registrazione Creazione Utente – Diagramma di Sequenza



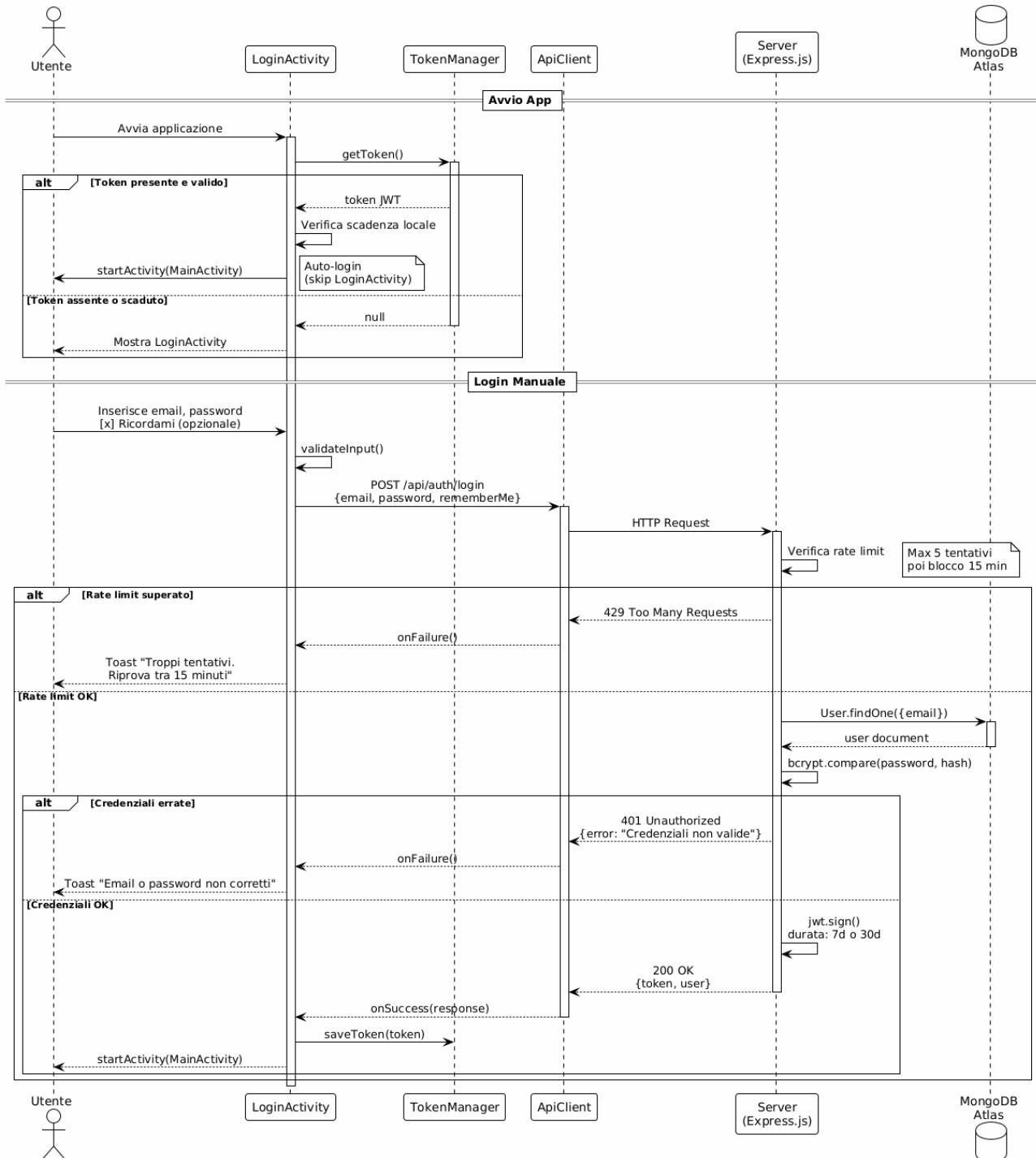
### Registrazione Self-Service EcoApp



### Login – Diagramma di Sequenza



## Login EcoApp



## Attributi di qualità della progettazione



Le scelte architetturali effettuate puntano a massimizzare alcuni attributi di qualità, come illustrato nelle slide:

- **Modificabilità**

L'alta coesione dei moduli (Auth, Profilo, CacheLocale, APIService) e il basso accoppiamento verso la logica server semplificano l'evoluzione futura del sistema.

- **Affidabilità**

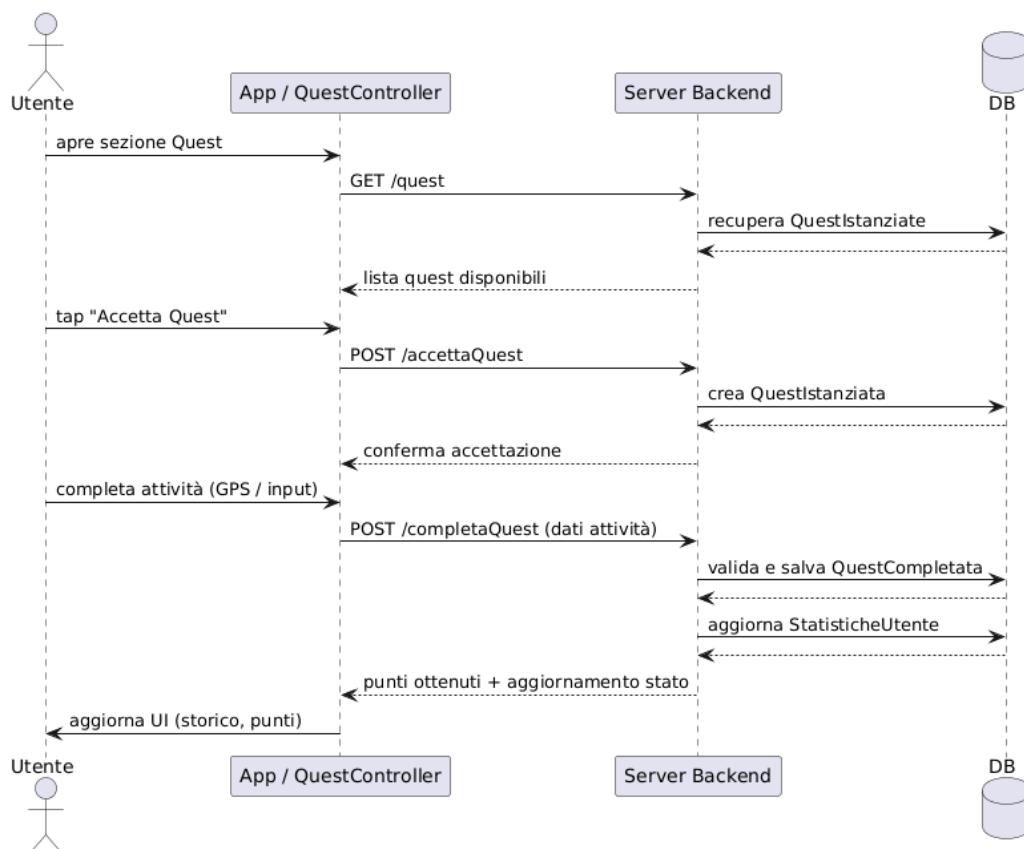
L'uso della cache locale e il fallback offline riducono la dipendenza dalla rete, con effetto positivo sull'esperienza utente e sulla resilienza del sistema.

- **Riutilizzabilità**

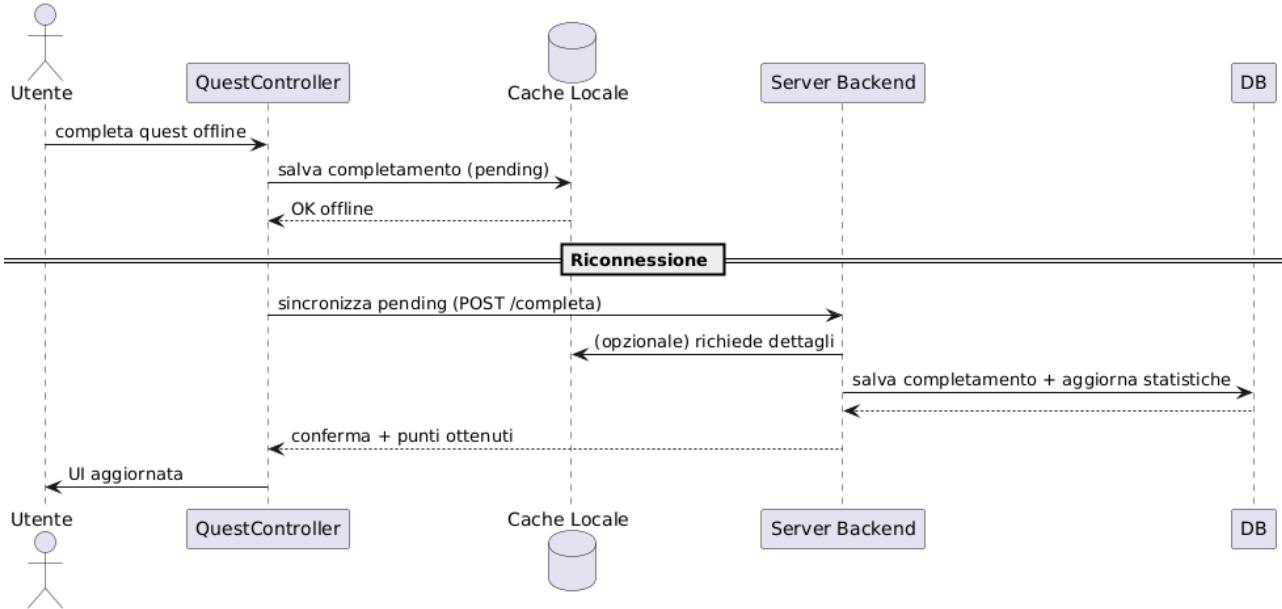
Controller, viste e modelli di dominio sono progettati come componenti indipendenti e riutilizzabili.

## Quest

### Diagramma Flusso Principale - Diagramma di sequenza



### Diagramma Modalità Offline - Diagramma di sequenza



## 5. Interfaccia utente



### Profilo

- Header (Identità e Riconoscimento):** Presenta l'avatar dell'utente, il **nickname** e il nome reale in modo gerarchico. Include il titolo del livello e il tasto **MODIFICA**, separando la gestione dell'account dalle metriche attive.
- Dashboard Impatto (Statistiche):** Questa sezione centrale traduce i dati di **StatisticheUtente** in feedback visivo immediato. I contatori dei **Punti Totali** e della **CO<sub>2</sub> Salvata** quantificano l'impatto ambientale, mentre l'indicatore circolare di progresso stimola il completamento del livello attuale.
- Sezione Social e Badge (Riconoscimenti):** Gestisce le interazioni e i traguardi raggiunti. Il box **I Miei Amici** fornisce un accesso rapido alla rete sociale, mentre la riga **I Miei Badge** mostra icone colorate che rappresentano i riconoscimenti sbloccati, offrendo una gratificazione visiva per gli obiettivi raggiunti.
- Sistema di Navigazione:** La **Bottom Navigation Bar** mantiene il contesto globale dell'app, evidenziando l'icona "Profilo" come sezione attiva. Questo garantisce che l'utente possa passare fluidamente dallo storico delle proprie azioni alla gestione delle nuove **Quest**.



## Registrazione / Creazione Utente

### Descrizione funzionale

#### Elementi principali:

- **Header:** "Crea il tuo account" con logo EcoApp
  - **Form:**
    - Campo Nome (obbligatorio)
    - Campo Email con validazione real-time
    - Campo Password con toggle visibilità
    - Campo Conferma Password
- **Call to action:**
  - Pulsante "Registrati" (abilitato solo se form valido)
  - Link "Hai già un account? Accedi"
- **Feedback post-registrazione:**
  - Toast: "Benvenuto [nome]!" con animazione confetti
  - Redirect automatico a MainActivity (2 secondi)
  - NO messaggio "controlla la mail" (account già attivo)

## Login

### Descrizione funzionale

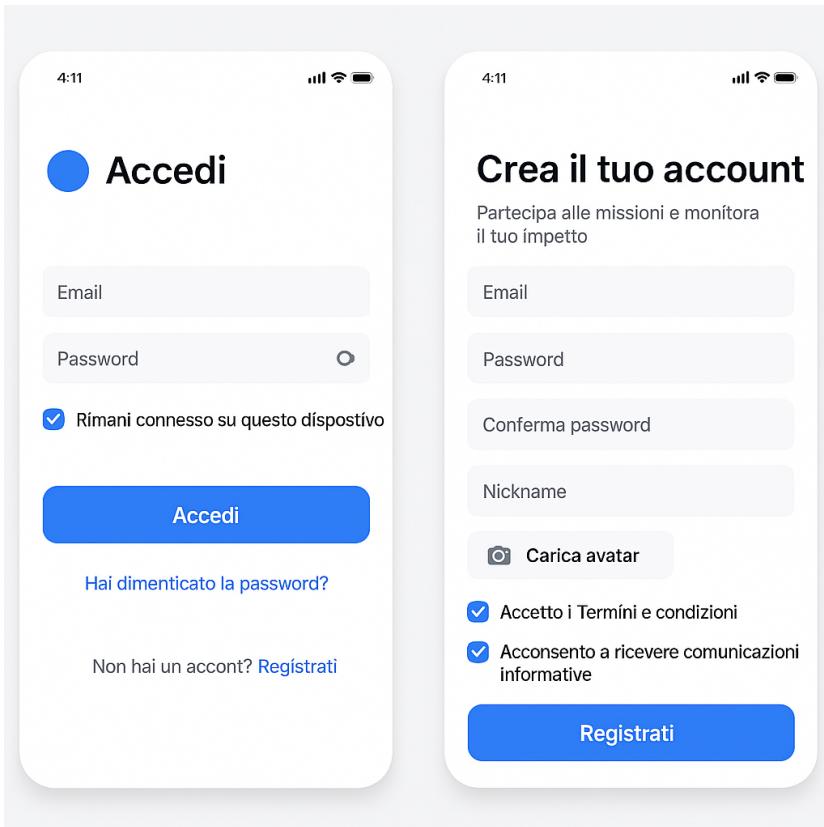
La schermata di login è pensata per essere molto essenziale e rapida, minimizzando gli ostacoli all'accesso quotidiano.

#### Elementi principali:

- **Header**
  - Logo / nome dell'app.
  - Titolo "Accedi".
- **Form**
  - Campo *Email*
  - Campo *Password* con opzione mostra/nascondi.
  - Checkbox "Rimani connesso su questo dispositivo".
- **Azioni principali**
  - Pulsante "Accedi"
  -



- Link "Non hai un account? Registrati" che rimanda alla schermata di registrazione.
- **Gestione stati**
  - Indicatore di caricamento (spinner) durante la chiamata al backend.
  - Messaggi di errore chiari in caso di credenziali errate o account non verificato.
  - Messaggio dedicato in caso di mancanza di rete ("Connessione assente. Controlla la rete e riprova.").



Mockup generato con ChatGPT

## Quest

### Descrizione Funzionale

La sezione *Quest* permette all'utente di visualizzare, accettare e completare missioni giornaliere e settimanali, con feedback immediato sul progresso e sui punti ottenuti. La struttura privilegia chiarezza, gamification e rapidità d'uso.

Elementi principali:

- **Header**
  - Titolo ("Quest")
  - Tab (Globali, In corso, Completate).
  - Barra di ricerca(cerca tra i titoli e le tipologie delle quest).



- **Lista Quest Globali (Core della schermata):** Le quest vengono mostrate in forma di card compatte. Sono visibili solo le quest mai toccate dall’utente.
  - Immagine Quest.
  - Titolo.
  - Tipologia.
  - Punteggio base.
- **Dettagli Quest Globale:**
  - Immagine Quest.
  - Titolo.
  - Tipologia.
  - Punteggio base.
  - CO2 risparmiata.
  - Descrizione
  - Obiettivi EU
  - Pulsanti contestuali:
    - Accetta Quest
- **Lista Quest in corso**
  - Immagine Quest.
  - Titolo.
  - Tipologia
  - Punteggio base.
- **Dettagli Quest in corso:**
  - Immagine Quest.
  - Titolo.
  - Tipologia.
  - Punteggio base.
  - CO2 risparmiata.
  - Descrizione
  - Obiettivi EU
  - Pulsanti contestuali:
    - Accetta Quest
    - Abbandona Quest
    - -1
    - +1
    - -10
    - +10
- **Lista Quest completate**
  - Immagine Quest.
  - Titolo.
  - Tipologia
  - Punteggio base.
  - Ripetizioni
  -



- **Dettagli Quest Completate:**

- Immagine Quest.
- Titolo.
- Tipologia.
- Ripetizioni.
- Punteggio per la singola quest.
- Punteggio totale.
- CO2 risparmiata per la singola quest.
- CO2 risparmiata in totale.
- Descrizione
- Obiettivi EU
- Pulsanti contestuali:
  - Ricomincia quest

- **Gestione attività e completamento** (Quando l'utente seleziona una quest)

- Apertura scheda dedicata con condizioni di completamento, punti, scadenza, progressi, e bottoni d'azione.
- Per quest manuali: pulsante *Conferma* (con eventuale allegato foto se necessario).

- **Gestione offline**

- Le quest accettate rimangono disponibili in locale.
- I completamenti restano “In attesa”.
- Sincronizzazione automatica alla riconnessione con toast di conferma.

- **Navigazione:** La bottom bar mette in evidenza la scheda attuale “Quest”, garantendo transizione rapida.

- Quest.
- Profilo.
- Settings.



## MockUp delle quest globali e dettagli quest globali

The image shows two side-by-side screenshots of a mobile application interface, likely for a rewards or fitness app.

**Left Screenshot (Quest List):**

- Top bar: "Quest" with a back arrow.
- Section header: "Globali" (underlined), "In corso", "Completate".
- Search bar: "Cerca per nome o tipo..." with a magnifying glass icon.
- Two thumbnail images: one of a cyclist and one of a runner.
- Card for the current quest:
  - Esploratore a Impatto Zero** (Mobilità)
  - 1000 pts**
- Bottom navigation: "Quest", "Profilo", "Settings".

**Right Screenshot (Quest Details):**

- Top bar: "Dettagli" with a back arrow.
- Large image: A cyclist in motion and a runner on a bridge.
- Section header: **Esploratore a Impatto Zero** (Mobilità).
- Metrics:
  - CO<sub>2</sub> RISPARMIATA **2,50 kg**
  - PUNTI PREMIO **1000 pts**
- Description:

Sostituisci l'auto con le tue gambe o la bicicletta per i tuoi spostamenti quotidiani fino
- Call-to-action button: **Accetta Quest**.
- Bottom navigation: "Quest", "Profilo", "Settings".



## MockUp delle ongoing quest e dettagli ongoing quest

23:28 23:29

Quest Dettagli

Globali In corso Completate

Cerca per nome o tipo...



**Custode dell'Oro Blu**

Risorse Idriche 300 pts

1 / 3

6 CLEAN WATER AND SANITATION 13 CLIMATE ACTION

ABBANDONA QUEST

1 / 3

-10 -1 +1 +10

Completa Quest

Quest Profilo Settings Quest Profilo Settings



## MockUp delle completed quest e dettagli completed quest

23:29

Quest

Globali In corso Completate

Cerca per nome o tipo...

**Idratazione Sostenibile**  
Consumo Responsabile **1000 pts**  
COMPLETATA 2 VOLTE

23:29

Dettagli

**COMPLETATA 2 VOLTE**

**Idratazione Sostenibile**  
*Consumo Responsabile*

CO<sub>2</sub> RISPARMIATA TOT. **1,60 kg** | PUNTI PREMIO TOT. **1000 pts**

CO<sub>2</sub> RISPARMIATA **0,80 kg** | PUNTI PREMIO **500 pts**

Ricomincia Quest

Quest Profilo Settings

Quest Profilo Settings