Progetto di Ingegneria del Software 2025/26

Università Ca' Foscari Venezia

Codice consegnato

1.0

EcoGroup

Document Informations

NomeProgetto		Acronimo	
Deliverable	D6		
Data di Consegna	22/12/2025		
Team Leader	Enrico Sforza		902600@stud.unive.it
Team members	Enrico	Rampazzo	901141@stud.unive.it
	Alberto Ferragosti 895936@stud.unive.it		

Document History

Version	Issue Date	Stage	Changes	Contributors
1.0	22/12/2025	Draft	Login e registrazione	895936
1.1	22/12/2025	Draft	Profilo	902600
2.0	22/12/2025	Final	Quest	901141
2.1	26/01/26	Final	Revisione pre consegna	895936, 902600

GITHUB

https://github.com/Enricoooh/ecoapp/tree/main

Indice

1. Login e registrazione ***4***

2. Profilo ***11***

2.1 Architettura Backend e Gamification 11

2.2 Modelli Dati (Backend e Frontend) 11

2.3 Logica e Interazione Frontend 12

2.4 Interfaccia Utente (UI/UX) 12

3. Quest ***12***

1. Panoramica 12

2. Architettura Android 13

2.1 Struttura del Package 13

2.2 Componenti Chiave 13

Quest.java (Modello Dati) 13

QuestFragment.java 13

QuestDetailFragment.java 13

QuestAdapter.java 13

3. Design UI e UX 14

3.1 Layout Features 14

3.2 Accessibilità e Localizzazione 14

4. Logica di Business e Flussi 14

5. Stato dello Sviluppo e Roadmap 14

5.1 Funzionalità Completate 14

5.2 Sviluppi Futuri (Backlog) 15

1. Login e registrazione

1. Panoramica

Implementato un sistema completo di autenticazione utente per l'applicazione Android EcoApp, comprendente registrazione, login e gestione della sessione. L'architettura segue il pattern client-server con comunicazione REST API e autenticazione basata su JWT

2. Architettura Backend

2.1 Stack Tecnologico
- Node.js 18+ con framework Express 4.18.2
- bcryptjs 2.4.3 per hashing sicuro delle password
- jsonwebtoken 9.0.2 per generazione e verifica token JWT
- CORS 2.8.5 per cross-origin requests
- Database: MongoDB Atlas via Mongoose
- Email Service: Resend per invio mail benvenuto

2.2 Struttura Backend

File principale: backend/server.js

Endpoint implementati:
- POST /api/auth/register - Registrazione nuovo utente
- POST /api/auth/login - Login con email/password
- GET /api/user/profile - Recupero dati profilo (protetto)
- PUT /api/user/profile - Aggiornamento profilo (protetto)

2.3 Sicurezza
- Password hashate con bcrypt (salt rounds: 10\)
- Token JWT: 7 giorni default, 30 giorni con opzione "Ricordami"
- Middleware authenticateToken per route protette
- Validazione input lato server (email formato, password min 6 caratteri)
- Rate limiting: 5 tentativi login falliti = blocco 15 minuti (express-rate-limit)

2.4 Deployment
- Hosting: Render.com (free tier)
- URL produzione: https://ecoapp-p5gp.onrender.com
- Configurazione: Root Directory = backend, Build = npm install, Start = npm start

3. Architettura Android

3.1 Package Struttura

Tutto il codice auth è isolato in com.ecoapp.android.auth/:
```
auth/
 ├── LoginActivity.java
 ├── RegisterActivity.java
 ├── AuthManager.java
 ├── TokenManager.java
 ├── AuthService.java (Retrofit interface)
 ├── ApiClient.java
 ├── AuthInterceptor.java
 └── models/
     ├── User.java
     ├── AuthResponse.java
     ├── LoginRequest.java
     ├── RegisterRequest.java
     └── ErrorResponse.java
```
3.2 Componenti Chiave
LoginActivity.java
- Gestisce il flusso di login con validazione input
- Layout: activity_login.xml (tema green gaming Duolingo-style)
- Validazioni: formato email, password min 6 caratteri
- Stati UI: loading spinner, error handling con Toast
RegisterActivity.java
- Registrazione con validazione completa (nome, email, password, conferma password)
- Layout: activity_register.xml
- Feature speciale: Animazione confetti con Konfetti library al successo registrazione
- Delay 2 secondi pre-navigazione per godersi l'animazione
AuthManager.java
Singleton per gestione stato autenticazione:
- isAuthenticated() - Verifica presenza token
- saveAuthData(token, user) - Salva credenziali post-login/register
- logout() - Cancella tutti i dati salvati
- getCurrentUser() - Recupera oggetto User corrente
TokenManager.java
Wrapper SharedPreferences per persistenza dati:
- Chiavi: jwt_token, user_id, user_email, user_name
- Preference name: "EcoAppPrefs"
- Metodi CRUD: save/get/clear per ciascun campo

ApiClient.java
Configurazione Retrofit centralizzata:
- BASE_URL: https://ecoapp-p5gp.onrender.com
- OkHttpClient con 30s timeout
- Interceptors: AuthInterceptor + HttpLoggingInterceptor
- GsonConverterFactory per serializzazione JSON
AuthInterceptor.java
Interceptor OkHttp per automatic JWT injection:
- Legge token da TokenManager
- Aggiunge header Authorization: Bearer {token} ad ogni request
- Permette chiamate senza token se non presente (per register/login)
AuthService.java
Retrofit Interface con endpoints tipizzati:
- register(@Body RegisterRequest) → Call<AuthResponse>
- login(@Body LoginRequest) → Call<AuthResponse>
- getProfile() → Call<User> (JWT automatico via interceptor)

4. Flusso di Autenticazione
4.1 Registrazione
1. User compila form in RegisterActivity
2. Validazione locale: nome min 2 char, email formato valido, password min 6 char, password match
3. POST /api/auth/register con RegisterRequest(email, password, name)
4. Backend: hash password (bcrypt), salva in MongoDB via User.create(), genera JWT
5. Backend: invio email di benvenuto tramite Resend service (fire-and-forget)
6. Response: AuthResponse con token \+ dati user
7. Android: AuthManager.saveAuthData() → SharedPreferences
8. Animazione confetti (Konfetti library, colori eco: \#58CC02, \#1CB0F6, \#FFC800, \#FF9600)

4.2 Login
1. User inserisce email/password in LoginActivity
2. Validazione locale: email formato, password min 6 char
3. POST /api/auth/login con LoginRequest(email, password)
4. Backend: lookup user, bcrypt compare password, genera JWT se match
5. Response: AuthResponse con token + dati user
6. Android: AuthManager.saveAuthData() → SharedPreferences
7. Navigazione a MainActivity (flags: NEW_TASK | CLEAR_TASK)
4.3 Persistent Login
MainActivity.java (righe 26-34):

```java
AuthManager authManager = AuthManager.getInstance(this);
if (!authManager.isAuthenticated()) {
    Intent intent = new Intent(this, LoginActivity.class);
    startActivity(intent);
    finish();
    return;
}
```

Comportamento: se token presente in SharedPreferences, skip login e carica app direttamente. Login richiesto solo se token assente.

4.4 Logout

SettingsFragment.java:

- Bottone "Esci dall'account"
- AuthManager.logout() → TokenManager.clearAll() → SharedPreferences.clear()
- Navigazione a LoginActivity (flags: NEW_TASK | CLEAR_TASK)
- Toast conferma

5. Integrazioni nell'App Esistente

5.1 MainActivity

Modifiche minime (8 righe aggiunte in onCreate()):

- Auth check all'avvio
- Redirect a LoginActivity se non autenticato
- Non toccato: QuestFragment, FriendsFragment (coordinamento team)

5.2 ProfileFragment

ProfileFragment.java:

- Integrato loadUserProfile() con chiamata GET /api/user/profile
- Sostituite stringhe hardcoded con dati reali da backend
- Update UI: nome utente, livello, punti totali, CO2 risparmiata (con Locale formatting)

5.3 SettingsFragment

SettingsFragment.java:

- Aggiunto MaterialButton "Esci dall'account" (rosso)
- Click listener → performLogout() → AuthManager + navigazione

6. Design UI

6.1 Palette Colori

app/src/main/res/values/colors.xml:

```xml
<color name="eco_green_primary">#58CC02</color>
<color name="eco_blue">#1CB0F6</color>
<color name="eco_yellow">#FFC800</color>
<color name="eco_orange">#FF9600</color>
<color name="eco_background">#F0F8F0</color>
```

Ispirazione: Duolingo gaming styl3.

6.2 Layout Features

- MaterialCardView con corner radius 16dp per input fields
- Emoji nei placeholder (👤 📧 🔒 🎉 🚀)
- Bottoni 64dp altezza, corner radius 32dp, elevation 8dp
- ScrollView per supporto landscape/tastiere
- ProgressBar overlay durante chiamate API
- Placeholder ImageView per futura mascotte eco

6.3 Animazione Confetti

RegisterActivity.java (metodo celebrateWithConfetti):

```java
EmitterConfig emitterConfig = new Emitter(5L, TimeUnit.SECONDS).max(100);
Party party = new PartyFactory(emitterConfig)
    .shapes(Arrays.asList(Shape.Circle.INSTANCE, Shape.Square.INSTANCE))
    .colors(Arrays.asList(0xFF58CC02, 0xFF1CB0F6, 0xFFFFC800, 0xFFFF9600))
    .setSpeedBetween(0f, 30f)
    .position(0.5, 0.0, 1.0, 0.0)
    .build();
binding.konfettiView.start(party);
```

Dependency: nl.dionsegijn:konfetti-xml:2.0.4 in app/build.gradle.kts

7. Dipendenze Aggiunte

app/build.gradle.kts:

```kotlin
// Networking
implementation("com.squareup.retrofit2:retrofit:2.9.0")
implementation("com.squareup.retrofit2:converter-gson:2.9.0")
implementation("com.squareup.okhttp3:okhttp:4.12.0")
implementation("com.squareup.okhttp3:logging-interceptor:4.12.0")
implementation("com.google.code.gson:gson:2.10.1")
```

// Confetti animation
implementation("nl.dionsegijn:konfetti-xml:2.0.4")
```

## 8. Permessi Manifest

app/src/main/AndroidManifest.xml:
```xml
<uses-permission android:name="android.permission.INTERNET" />
<activity android:name="com.ecoapp.android.auth.LoginActivity" />
<activity android:name="com.ecoapp.android.auth.RegisterActivity" />
```

9. Testing & Debugging

9.1 Logging
- OkHttp HttpLoggingInterceptor: BODY level per debug request/response
- Toast messages: feedback immediato per user su errori
- Console.log backend: tutti gli errori catturati

9.2 Error Handling
- Network failures: Callback onFailure() con t.getMessage() in Toast
- API errors: Parse ErrorResponse da response.errorBody() con Gson
- Validation errors: TextInputLayout.setError() per feedback inline

9.3 Edge Cases Gestiti
- Email già registrata → 409 Conflict con messaggio specifico
- Credenziali errate → 401 con messaggio generico (sicurezza)
- Token scaduto/invalido → 403 da middleware backend
- Render.com cold start → timeout 30s configurato (free tier richiede 60s primo boot)

10. Decisioni Architetturali

10.1 JWT
- Stateless: no session storage server-side (scalabilità)
- Mobile-friendly: token salvato localmente, no cookies
- Expiration built-in: 30 giorni automatico

10.2 SharedPreferences
- Semplicità: no database locale necessario per pochi dati
- Performance: accesso istantaneo in-memory
- Security nota: dati plain text (OK per MVP, EncryptedSharedPreferences per produzione)

10.3 Singleton Pattern
- AuthManager e ApiClient evitano istanze multiple
- Garantisce stato consistente app-wide
- Context leak prevention con context.getApplicationContext()
```

10.4 File JSON Backend
- No dipendenze: no MongoDB/PostgreSQL setup
- Sufficiente per MVP: Render.com persistence disk, scalabile in futuro

11. Limitazioni & Future Enhancements

11.1 Limitazioni Attuali
- Database: JSON non scalabile, no concurrent writes protection
- Security: JWT_SECRET hardcoded (OK per dev, variabile env per prod)
- SharedPreferences: plain text (upgrade a EncryptedSharedPreferences raccomandato)
- Email verification: non implementata (scelta consapevole per semplicità MVP)
- Token refresh: no automatic refresh (user deve rifare login dopo 30gg)

11.2 Possibili Upgrade
- Email verification (attivazione account via link)
- Token refresh endpoint
- Social login (Google/Facebook)
- Biometric authentication (Fingerprint/Face ID)
- Custom font Nunito (menzionato ma non implementato)

13. Deployment Notes

13.1 Render.com Configuration
- Root Directory: backend (monorepo structure)
- Build Command: npm install
- Start Command: npm start
- Auto-deploy: GitHub push su main branch
- Cold start: ~50s first request dopo inattività (free tier)

13.2 Android Configuration
app/src/main/java/com/ecoapp/android/auth/ApiClient.java:
```java
private static final String BASE_URL = "https://ecoapp-p5gp.onrender.com";
```
Cambiare per ambienti diversi (localhost, staging, prod).

14. Conclusioni

Il sistema implementato fornisce una base solida e sicura per l'autenticazione utente in EcoApp, rispettando i requisiti di semplicità e rapidità. L'architettura modulare permette facile estensione futura senza impattare il lavoro dei colleghi del team. Il design green gaming e l'animazione confetti aggiungono un tocco ludico coerente con la mission eco-friendly dell'app.

2. Profilo

Il modulo **Profilo** rappresenta la dashboard personale dell'utente all'interno di EcoApp. È il punto dove i dati di autenticazione si fondono con la logica di *gamification* e il tracciamento dell'impatto ambientale. Il presente documento descrive l'architettura *full-stack* implementata per questo modulo.

2.1 Architettura Backend e Gamification

server.js (Node.js/Express):
Il backend gestisce tutte le rotte API relative al profilo utente, inclusa l'autenticazione. Le sue responsabilità principali includono:

- **Gestione Rotte:** Endpoint protetti (`GET /api/user/profile`, `PUT /api/user/profile`) per la lettura e l'aggiornamento dei dati personali e delle statistiche.
- **Motore di Livellamento:** Logica per il calcolo dinamico del **livello utente** (es. da Eco-Novizio a Eco-Leggenda) basato sui `reward_points` accumulati.
- **Sistema Badge:** Implementazione del `checkBadges` che sblocca automaticamente i riconoscimenti digitali in base ai punti totali e ai dati ambientali tracciati (CO2 risparmiata, missioni completate).

2.2 Modelli Dati (Backend e Frontend)

User.java (Modello Frontend):
È il *Plain Old Java Object (POJO)* che rispecchia l'entità utente nel database, garantendo la coerenza dei dati tra server e client. Trasporta informazioni essenziali quali:

- Dati Anagrafici (email, nickname).
- Statistiche Ambientali (CO2 risparmiata, punti totali).
- Metadati di Gioco (livello corrente, lista `Badge.java`, lista amici).

Badge.java (Modello Frontend):
Gestisce la rappresentazione dei riconoscimenti sbloccati. Contiene la funzione cruciale

`getImageResId()`, che risolve l'associazione tra l'ID del badge ricevuto via API (stringa) e la risorsa grafica (`drawable`) corrispondente nell'applicazione Android.

2.3 Logica e Interazione Frontend

ProfileFragment.java:
È il controller principale per la visualizzazione del profilo.

- **Integrazione API:** Effettua chiamate `GET /api/user/profile` tramite Retrofit per recuperare i dati utente in tempo reale.
- **Aggiornamento UI:** Popola dinamicamente i widget del layout con i dati provenienti dal backend (nome, livello, statistiche).
- **Barra di Progresso:** Contiene la logica `calculateLevelPercentage` che determina la progressione visiva sul `CircularProgressIndicator` per motivare l'utente al livello successivo.

2.4 Interfaccia Utente (UI/UX)

- **fragment_profile.xml (Dashboard):**
  - Layout pulito e moderno, basato su **MaterialCardView** con design a schede.
  - Elementi chiave: **CircularProgressIndicator** per il livello, **RecyclerView orizzontale** per visualizzare i badge sbloccati.
- **fragment_edit_profile.xml (Personalizzazione):**
  - Interfaccia dedicata alla modifica dei dati utente (es. nickname).
  - Utilizzo di **TextInputLayout** per l'input guidato e **ImageView circolare** per l'anteprima dell'immagine profilo.
- **fragment_friends.xml (Social):**
  - Sezione per la gestione delle interazioni sociali.
  - Include una barra di ricerca e due **RecyclerView** per separare chiaramente le richieste di amicizia in sospeso dalla lista amici consolidata.

3. Quest

1. Panoramica

Il modulo Quest costituisce il cuore dell'esperienza di *gamification* di EcoApp. È stato progettato per incentivare comportamenti eco-sostenibili attraverso un sistema di missioni monitorabili. Il sistema permette la visualizzazione di una lista di sfide attive e l'approfondimento di ciascuna di esse tramite una vista di dettaglio che integra obiettivi, ricompense e tracciamento del progresso in tempo reale.

2. Architettura Android

2.1 Struttura del Package

Il codice relativo alle missioni è organizzato nel package `com.example.ecoapp.quest_files/`, mantenendo una netta separazione tra la logica dei dati (modelli) e la logica di presentazione:

```
quest_files/
├── QuestFragment.java        (Visualizzazione lista)
├── QuestDetailFragment.java  (Visualizzazione dettaglio)
├── QuestAdapter.java         (Bridge dati-UI per RecyclerView)
└── models/
    └── Quest.java            (Modello dati POJO)
```

2.2 Componenti Chiave

Quest.java (Modello Dati)

Rappresenta l'entità fondamentale della missione. Include attributi per:

- **Identificazione**: `id`, `name`, `type` (es. Alimentation, Mobility).
- **Gamification**: `reward_points`, `actual_progress`, `max_progress`.
- **Contenuto**: `description`, `imageResId`.
- **Sostenibilità**: `images_eu_goals` (array di riferimenti alle icone degli obiettivi ONU 2030).

QuestFragment.java

- Gestisce la visualizzazione principale delle missioni disponibili.
- Utilizza una `RecyclerView` per garantire performance elevate nello scrolling di liste potenzialmente lunghe.
- Implementa la logica di navigazione verso il dettaglio tramite `Navigation Component`, passando l'ID della quest selezionata come argomento.

QuestDetailFragment.java

- Implementa una visualizzazione gerarchica avanzata:
  - **Top**: Immagine rappresentativa della missione.
  - **Center**: Corpo informativo scorrevole (`NestedScrollView`) contenente nome, descrizione estesa e tipologia.
  - **Social Impact**: Sezione dinamica dedicata alle icone degli obiettivi EU Sustainable Development Goals.
  - **Fixed Bottom**: Una barra di progresso persistente ancorata al fondo della schermata per mantenere sempre visibile l'avanzamento dell'utente.

QuestAdapter.java

- Implementa il pattern `ViewHolder` per il riciclo efficiente delle viste.
- Gestisce il binding dinamico dei dati, inclusa la configurazione delle `ProgressBar` e delle immagini di anteprima.
- Definisce l'interfaccia `OnQuestClick` per il disaccoppiamento della logica di click.

3. Design UI e UX

3.1 Layout Features

- **Item Quest (`item_quest.xml`)**: Utilizzo di `MaterialCardView` con corner radius di 16dp ed elevazione di 4dp per un look moderno e "pulito".
- **Dettaglio Quest (`fragment_quest_detail.xml`)**:
  - Layout basato su `ConstraintLayout`.
  - Immagini in `scaleType="centerCrop"` per una resa visiva cinematografica.
  - Utilizzo di colori semantici (es. `#4CAF50` per i Reward Points).

3.2 Accessibilità e Localizzazione

- **Content Descriptions**: Gestione degli attributi `contentDescription` (settaggio a `@null` per elementi decorativi) per garantire la compatibilità con TalkBack.
- **Resource Strings**: Utilizzo di placeholder dinamici nel file `strings.xml` per evitare la concatenazione di stringhe nel codice Java (es. `reward_label`).

XML

```
<string name="reward_label">Reward: %1$d coins</string>
<string name="quest_image_desc">Immagine della quest</string>
```

4. Logica di Business e Flussi

1. **Inizializzazione**: All'apertura del `QuestFragment`, viene generata una lista di oggetti `Quest` (attualmente via mock data, predisposta per fetch API).
2. **Selezione**: Al click su una card, l'adapter comunica l'ID al fragment padre.
3. **Navigazione**: Il `NavController` esegue la transizione verso `QuestDetailFragment`.
4. **Rendering Dettaglio**: Il fragment di dettaglio riceve l'ID, recupera l'istanza della Quest e popola dinamicamente i campi. Le icone degli obiettivi EU Goals vengono generate a runtime e inserite in un contenitore orizzontale.

5. Stato dello Sviluppo e Roadmap

5.1 Funzionalità Completate

- [x] Struttura dati `Quest` con supporto per obiettivi ONU 2030.
- [x] Visualizzazione a lista con `RecyclerView` e `CardView`.
- [x] Navigazione sicura tra lista e dettaglio.
- [x] Layout dettaglio con barra di progresso fissa e scrolling intelligente.

5.2 Sviluppi Futuri (Backlog)

- **Integrazione Backend**: Sostituzione della lista locale con chiamate `GET /api/quests` tramite Retrofit.
- **Update Progresso**: Implementazione di pulsanti di azione nel dettaglio per aggiornare lo stato di avanzamento della missione (chiamate `POST`).
- **Filtri**: Aggiunta di chip per filtrare le missioni per categoria (es. solo "Mobility").
- **Mascotte**: Integrazione di animazioni Lottie per celebrare il completamento di una missione.