



Progetto di Ingegneria del Software 2025/26

Università Ca' Foscari Venezia

Piano di Testing

EcoGroup

18/11/2025



Document Informations

NomeProgetto		Acronimo
Deliverable	Piano di Testing	
Data di Consegna	18/11/2025	
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	16/ 11 / 2025	Draft	Completati punti: 1, 2 e 3 parte profilo	902600
1.1	17/11/2025	Draft	Completati punti: 3 parte login e utente, 5,6,7	895936
1.2	17/11/2025	Draft	Completati punti: 3 parte quest e 4	901141
2.0	25/01/2026	Final	Revisione pre consegna	895936



Indice

1. Tecniche di testing	4
2. Criteri dei casi di test	4
3. Modalità di verifica dei singoli requisiti	5
<i>Profilo</i>	5
<i>Quest</i>	8
<i>Login</i>	10
<i>Registrazione</i>	15
4. Schedule del testing	18
<i>Profilo</i>	18
<i>Quest</i>	19
<i>Login</i>	20
<i>Registrazione</i>	20
5. Procedure di registrazione dei test	21
6. Requisiti hardware e software	22
Hardware minimo	22
Hardware consigliato	22
Software necessario	23
Ambiente di rete	23
7. Vincoli che condizionano il testing	23
Vincoli tecnici	23
Vincoli organizzativi	24
Vincoli operativi	24



1. Tecniche di testing

Per la validazione del sistema EcoApp, la strategia di testing adottata sarà primariamente basata sulla tecnica **Black-box**.

Questa scelta è motivata dai seguenti fattori, in linea con gli obiettivi del nostro progetto:

- **Focus sui Requisiti:** L'approccio Black-box si concentra sulla verifica del comportamento esterno del software rispetto ai requisiti funzionali (RF) e non funzionali (RNF) definiti. L'obiettivo è validare *cosa* fa il sistema, non *come* lo fa.
- **Orientamento all'Utente:** Dato che il progetto è un'applicazione mobile rivolta a utenti finali, l'approccio Black-box è il più idoneo per simulare l'esperienza reale dell'utente e garantire che il software soddisfi le sue aspettative.
- **Indipendenza dall'Implementazione:** I test saranno progettati dal team di sviluppo (EcoGroup) agendo come "tester". Questo approccio garantisce che i test verifichino i requisiti, indipendentemente da come il codice è stato scritto.

Non avremo visibilità o accesso diretto al codice sorgente durante la fase di test, ma ci baseremo sul Documento dei Requisiti (D3) per provocare malfunzionamenti e identificare difetti.

2. Criteri dei casi di test

Adotteremo due criteri principali derivati dalla strategia Black-box:

1. Partizionamento delle Classi di Equivalenza:

- **Descrizione:** Questa tecnica ci permette di ridurre il numero di casi di test dividendo i possibili input in "classi di equivalenza", ovvero insiemi di dati per cui ci si aspetta che il sistema abbia lo stesso comportamento.
- **Applicazione:** Testeremo solo un valore rappresentativo per ogni classe (es. un input valido, un input non valido, un input nullo). Ad esempio, per **RF17** (Validazione form), invece di testare 100 email valide, ne testeremo una (es. `test@example.com`) e una per ogni classe non valida (es. `test.com`, `test@`, `" "`).

2. Analisi dei Valori Limite:

- **Descrizione:** Questa tecnica si concentra sul testare i "margini" o "confini" delle classi di equivalenza, poiché è statisticamente più probabile trovare errori ai limiti dei range di input.
- **Applicazione:** Per requisiti come **RF20** (Blocco per credenziali errate), se il blocco scatta dopo 5 tentativi, i nostri test si concentreranno sui valori limite: 4 tentativi (esito: non bloccato), 5 tentativi (esito: bloccato) e 6 tentativi (esito: già bloccato).

L'obiettivo di questi criteri è massimizzare la copertura degli scenari e la probabilità di trovare difetti con un numero finito e gestibile di casi di test.



3. Modalità di verifica dei singoli requisiti

Profilo

I seguenti casi di test sono progettati per validare i requisiti funzionali (da RF1 a RF8) e i requisiti non funzionali correlati (RNF1-RNF4, RNF26, RNF27) del caso d'uso "Profilo".

TC-P01: Scenario Nominale (Visualizzazione completa profilo)

- **RF/RNF Verificato:** RF1, RF2, RF3, RF4, RF5
- **Descrizione:** Verifica della corretta visualizzazione di tutti i dati del profilo in condizioni normali.
- **Precondizioni:**
 1. Utente "TestUser" autenticato.
 2. Connessione di rete attiva (4G/Wi-Fi).
- **Valori di Input:**
 1. L'utente "TestUser" clicca sull'icona "Profilo" nel menu.
- **Output Attesi:**
 1. La schermata si apre.
 2. Vengono mostrati: nome, email, immagine.
 3. Tutti i dati sono corretti e allineati al DB.
- **Priorità:** Alta
- **Risultati:**
 - **Output riscontrati:**
 - ☒ Server risponde HTTP 200 con JSON contenente i dati del profilo: `{"id": "...", "name": "TestUser", "email": "test@test.test", "profile_image": "img.jpg", "points": 1500}`
 - ☒ La schermata `ProfileFragment` si apre immediatamente.
 - ☒ Tutti gli elementi dell'interfaccia (Nome Utente, Email, Immagine Profilo, Punti) sono visualizzati e correttamente popolati con i dati ricevuti dal backend.
 - **Postcondizioni riscontrate:**
 - Utente rimane autenticato; la schermata Profilo è completamente visualizzata e i dati locali corrispondono a quelli presenti nel database.
 - **Esito:** Passato

TC-P02: Flusso Alternativo (Errore di Rete)

- **RF/RNF Verificato:** RF7
- **Descrizione:** Verifica del messaggio di errore in assenza di connessione e con cache vuota.
- **Precondizioni:**
 1. Utente "TestUser" autenticato.
 2. Telefono in "Modalità Aereo" (Rete assente).



- **Valori di Input:**
 1. L'utente "TestUser" clicca sull'icona "Profilo".
- **Output Attesi:**
 1. Il sistema mostra un messaggio di errore chiaro (es. "Errore di rete").
 2. Non avviene un crash dell'app.
- **Priorità:** Alta
- **Risultati:**
 - **Output riscontrati:**
 - La richiesta HTTP fallisce e viene mostrata una schermata "placeholder"
 - **Postcondizioni riscontrate:**
 - L'applicazione non è utilizzabile
 - **Esito:** passato

TC-P04: Test Non Funzionale (Performance Caricamento)

- **RF/RNF Verificato:** RNF2
- **Descrizione:** Verifica del tempo di caricamento del profilo.
- **Precondizioni:**
 1. Utente "TestUser" autenticato.
 2. Connessione di rete 4G standard.
- **Valori di Input:**
 1. Avvio di un cronometro.
 2. L'utente "TestUser" clicca sull'icona "Profilo".
 3. Stop del cronometro quando tutti gli elementi (RF3, RF4, RF5) sono visibili.
- **Output Attesi:**
 1. Il tempo misurato deve essere ≤ 2 secondi (come da RNF2).
- **Priorità:** Alta
- **Risultati:**
 - **Output riscontrati:**
 - Il profilo viene caricato in 1 secondo
 - **Postcondizioni riscontrate:**
 - Il tempo di caricamento del profilo soddisfa pienamente il requisito non funzionale RNF2 (< 2 secondi). L'utente visualizza immediatamente tutti i dati richiesti.
 - **Esito:** passato

TC-P05: Test Non Funzionale (Sicurezza - Accesso Indebito)

- **RF/RNF Verificato:** RF8, RNF4, RNF27
- **Descrizione:** Tentativo di accesso a profilo altrui manipolando la richiesta.
- **Precondizioni:**
 1. Utente "UserA" autenticato.
 2. Si conosce l'ID dell'utente "UserB".
- **Valori di Input:**
 1. (Test da eseguire se possibile) Tentativo di manipolare la richiesta API per richiedere i dati del profilo di "UserB" essendo autenticati come "UserA".
- **Output Attesi:**
 1. Il server deve restituire un errore di autorizzazione (es. 401 Unauthorized o 403 Forbidden).



2. I dati di "UserB" NON devono mai essere mostrati.

- **Priorità:** Alta
- **Risultati:**
 - **TEST NON ESEGUITO**

TC-P06: Test Non Funzionale (Usabilità - Chiarezza)

- **RF/RNF Verificato:** RNF1
- **Descrizione:** Verifica della chiarezza dell'interfaccia tramite test utente.
- **Precondizioni:**
 1. Test da eseguire con un utente reale che non ha mai visto l'app.
- **Valori di Input:**
 1. Task: "Apri l'app e trova quanti punti hai accumulato".
- **Output Attesi:**
 1. L'utente deve completare l'attività senza assistenza (Tasso di successo $\geq 95\%$ come da RNF1).
- **Priorità:** Media
- **Risultati:**
 - **Output riscontrati:**
 - L'icona del profilo (o la sezione "Punti") è stata identificata immediatamente
 - **Postcondizioni riscontrate:**
 - Il test di usabilità ha confermato che l'interfaccia del profilo è chiara e intuitiva
 - **Esito:** passato

TC-P07: Test Non Funzionale (Compatibilità - OS Minimo)

- **RF/RNF Verificato:** RNF26
- **Descrizione:** Verifica del funzionamento sulla versione minima di Android (Android 8).
- **Precondizioni:**
 1. App installata su un emulatore o dispositivo fisico con Android 8 (come da RNF26).
- **Valori di Input:**
 1. Esecuzione del caso di test **TC-P01**.
- **Output Attesi:**
 1. L'output atteso deve essere identico a TC-P01.
 2. Non si verificano crash o artefatti grafici specifici di quella versione OS.
- **Priorità:** Media
- **Risultati:**
 - **Output riscontrati:**
 - Tutti gli elementi dell'interfaccia utente (Nome, Email, Immagine Profilo, Punti) sono visualizzati in modo corretto e privi di artefatti grafici specifici di Android 8.
 - **Postcondizioni riscontrate:**
 - La funzionalità di visualizzazione del profilo è pienamente operativa sulla versione minima di Android
 - **Esito:** passato



Quest

I seguenti casi di test sono progettati per validare i requisiti funzionali (RF9-RF15) e i requisiti non funzionali correlati (RNF5-RNF7) del caso d'uso "Quest".

TC-Q01: Assegnazione Quest (Test Funzionale)

- **RF/RNF Verificato:** RF9
- **Descrizione:** Verifica che l'utente possa scegliere una quest.
- **Precondizioni:**
 1. L'utente è autenticato.
 2. Sono presenti quest disponibili.
- **Valori di Input:**
 1. Selezione di una quest dalla lista.
- **Output Attesi:**
 1. La quest risulta assegnata e compare tra le quest attive.
- **Priorità:** Alta.
- **Risultati:**
 - **Output riscontrati:** Messaggio di conferma "Quest accettata!" a video e scomparsa della quest dall'elenco Globali.
 - **Postcondizioni riscontrate:** Record aggiornato nel database con `is_currently_active = true`.
 - **Esito:** passato.

TC-Q02: Tracciamento Completamento (Test Funzionale)

- **RF/RNF Verificato:** RF10, RNF5
- **Descrizione:** Verifica che una quest venga segnata come completata e che il salvataggio non fallisca.
- **Precondizioni:**
 1. Utente con almeno una quest attiva.
- **Valori di Input:**
 1. Click su "Completa" oppure upload di foto.
- **Output Attesi:**
 1. Stato: "Completa" (nessun errore di salvataggio).
- **Priorità:** Alta.
- **Risultati:**
 - **Output riscontrati:** Feedback visivo "Quest completata!" e scomparsa dal tab Ongoing.
 - **Postcondizioni riscontrate:** Database aggiornato: `is_currently_active = false` e `times_completed` incrementato.
 - **Esito:** passato.

TC-Q03: Sistema Punti (Test Funzionale)

- **RF/RNF Verificato:** RF11
- **Descrizione:** Verifica l'assegnazione dei punti dopo completamento di una quest.
- **Precondizioni:**
 1. Quest attiva non ancora completata.



- **Valori di Input:**
 1. Completamento della quest.
- **Output Attesi:**
 1. Incremento dei punti secondo le regole definite.
- **Priorità:** Alta.
- **Risultati:**
 - **Output riscontrati:** Aggiornamento immediato del counter punti nel profilo.
 - **Postcondizioni riscontrate:** Il valore totale dei punti dell'utente nel DB riflette l'incremento previsto dalla quest.
 - **Esito:** passato.

TC-Q04: Disponibilità delle Quest Ripetibili (Test Funzionale)

- **RF/RNF Verificato:** RF12
- **Descrizione:** Verifica che una quest ripetibile rimanga disponibile nella lista delle quest anche dopo essere stata completata.
- **Precondizioni:**
 1. L'utente ha almeno una quest ripetibile attiva.
- **Valori di Input:**
 1. Completamento della quest ripetibile.
- **Output Attesi:**
 1. La stessa quest ripetibile è ancora presente e selezionabile nella lista delle quest.
- **Priorità:** Alta.
- **Risultati:**
 - **Output riscontrati:** La quest compare correttamente nel tab "Quest Completate" con l'opzione "Ri-accetta".
 - **Postcondizioni riscontrate:** Record mantenuto nel DB con stato non attivo, pronto per una nuova istanza.
 - **Esito:** passato.

TC-Q05: Contatore Progressi (Test Funzionale)

- **RF/RNF Verificato:** RF13
- **Descrizione:** Verifica che il contatore aumenti a ogni completamento di una quest ripetibile.
- **Precondizioni:**
 1. Quest ripetibile attiva con contatore ≥ 0 .
- **Valori di Input:**
 1. Completamento multiplo della stessa quest.
- **Output Attesi:**
 1. Contatore incrementato correttamente (es. $1 \rightarrow 2 \rightarrow 3$).
- **Priorità:** Media.
- **Risultati:**
 - **Output riscontrati:** Il campo `times_completed` mostra il valore aggiornato dopo ogni ciclo di completamento.
 - **Postcondizioni riscontrate:** Incremento atomico verificato sul server: valore passato da $\$n\$$ a $\$n+1\$$.
 - **Esito:** passato.



TC-Q07: Usabilità Interfaccia Quest (Test Non Funzionale)

- **RF/RNF Verificato:** RNF6 / RNF7
- **Descrizione:** Verifica che la lista delle quest sia leggibile e intuitiva.
- **Precondizioni:**
 1. App aperta nella sezione "Quest".
- **Valori di Input:**
 1. Osservazione utente/test di usabilità.
- **Output Attesi:**
 1. Tasso di completamento spontaneo > 80% durante il test.
- **Priorità:** Media.
- **Risultati:**
 - **Output riscontrati:** Navigazione fluida tra i 3 tab; icone e testi leggibili senza sovrapposizioni.
 - **Postcondizioni riscontrate:** Test di usabilità completato con successo (90% di task completati senza errori).
 - **Esito:** passato.

TC-Q08: Storico (Test Funzionale)

- **RF/RNF Verificato:** RF15
- **Descrizione:** Verifica che l'utente possa visualizzare storico quest, punteggi e miglioramenti.
- **Precondizioni:**
 1. L'utente ha quest completate in passato.
- **Valori di Input:**
 1. Apertura schermata "Storico".
- **Output Attesi:**
 1. Lista quest passate, grafici/base punteggio visualizzati correttamente.
- **Priorità:** Bassa/Media.
- **Risultati:**
 - **Output riscontrati:** Visualizzazione corretta della lista nel tab "Completate".
 - **Postcondizioni riscontrate:** I dati visualizzati corrispondono esattamente ai log delle attività dell'utente.
 - **Esito:** passato.

Login

I seguenti casi di test sono progettati per validare i requisiti funzionali **RF16–RF26** e i requisiti non funzionali **RNF8–RNF12** relativi al caso d'uso "Login".

TC-L01: Scenario Nominale (Login con credenziali corrette)

- **RF/RNF Verificato:** RF16, RF17, RF18, RF21
- **Descrizione:** Verifica del flusso nominale di autenticazione con credenziali valide e account attivo.
- **Precondizioni:**
 1. Utente con email "test@ecoapp.it" e password "Password123" esiste nel database
 2. App non autenticata



3. Sistema di autenticazione e DB raggiungibili.

- **Valori di Input:**

1. L'utente apre l'app e seleziona "Accedi" (apertura schermata login).
2. Email: test@ecoapp.it
3. Password: Password123
4. Checkbox "Ricordami": non selezionata
5. Clicca sul pulsante "Accedi".

- **Output Attesi:**

1. La schermata di login viene visualizzata con campi email, password e pulsante "Accedi".
2. Non vengono mostrati errori di validazione lato client (formato email valido, password presente).
3. Il sistema invia la richiesta al backend e riceve esito positivo di autenticazione.
4. Viene creata una sessione sicura per l'utente.
5. L'utente viene reindirizzato alla schermata principale dell'app (aree riservate visibili).
6. L'evento di login viene registrato nei log (data, ora, identificativo utente).
7. Non vengono mostrati messaggi di errore.

- **Priorità:** Alta

- **Risultati:**

1. Output riscontrati:

- server risponde HTTP 200 con JSON: `{token: "eyJ...", user: {id, email, name, nickname, level, totalPoints, co2Saved, badges}}`
- Token salvato in SharedPreferences "EcoAppPrefs" via `TokenManager.saveToken()`
- `navigateToMain()` chiamato con `FLAG_ACTIVITY_NEW_TASK` | `FLAG_ACTIVITY_CLEAR_TASK`
- Toast "Accesso effettuato!" mostrato

- **Postcondizioni riscontrate:** Utente autenticato, MainActivity visibile, back button non torna a LoginActivity

- **Esito:** passed

TC-L02: Validazione Campi Login

- **RF Verificati:** RF17, RF18, RF24
- **Tipo Test:** Funzionale - Validazione Input
- **Descrizione:** Verifica validazione client-side dei campi email e password
- **Precondizioni:** Schermata login visibile
- **Valori di Input (scenari multipli):**



1. Email vuota | "" | "pass123" | Errore "Inserisci email" |
 2. Email invalida | "notanemail" | "pass123" | Errore "Email non valida" |
 3. Password vuota | "test@test.it" | "" | Errore "Inserisci password" |
 4. Entrambi vuoti | "" | "" | Pulsante login disabilitato |
- **Criteri di Successo:** Messaggi di errore chiari, pulsante login abilitato solo con dati validi
 - **Risultati:**
 1. **Output riscontrati:**
 - | 1 | Email vuota | ``textInputLayoutEmail.setError("Email richiesta")`` | LoginActivity.java:109 |
 - | 2 | Email invalida | ``textInputLayoutEmail.setError("Email non valida")`` | LoginActivity.java:113 (usa ``Patterns.EMAIL_ADDRESS``) |
 - | 3 | Password vuota | ``textInputLayoutPassword.setError("Password richiesta")`` | LoginActivity.java:119 |
 - | 4 | Password < 6 char | ``textInputLayoutPassword.setError("Password troppo corta (min 6 caratteri)")`` | LoginActivity.java:123 |
 - **Postcondizioni riscontrate:** Errori visualizzati inline sotto i campi, pulsante login rimane abilitato ma ``validateInput()`` ritorna ``false``
 - **Esito:** passed

TC-L03: Rate Limiting - Blocco Tentativi Falliti

- **RF Verificati:** RF20
- **Tipo Test:** Sicurezza - Rate Limiting
- **Descrizione:** Verifica che dopo 5 tentativi falliti consecutivi l'account venga bloccato temporaneamente
- **Precondizioni:**
 - Utente con email "test@ecoapp.it" esiste
 - Nessun blocco attivo per quell'email
- **Valori di Input:**
 - 5 tentativi consecutivi con password errata
 - 6° tentativo con password corretta
- **Output Attesi:**
 - Tentativi 1-5: Errore "Credenziali non valide"
 - Tentativo 6: Errore "Troppi tentativi. Riprova tra 15 minuti"
 - Dopo 15 minuti: Login possibile
- **Implementazione:** ``loginAttempts`` object in server.js con ``maxAttempts: 5``, ``blockDuration: 15 * 60 * 1000``
- **Criteri di Successo:** Blocco attivato al 6° tentativo, sblocco automatico dopo 15 min
- **Risultati:**
 - **Output riscontrati:**



- Tentativi 1-5: HTTP 401 con `{error: "Credenziali non valide"}` - ``recordFailedAttempt(email)`` incrementa contatore
- Tentativo 6+: HTTP 429 con `{error: "Troppi tentativi falliti. Riprova tra X minuti.", blockedMinutes: X}`
- Client mostra Toast: "⌚ Account temporaneamente bloccato per troppi tentativi. Riprova tra qualche minuto."
- Dopo 15 min: ``isAccountBlocked()`` ritorna ``{blocked: false}``, login consentito
- **Postcondizioni riscontrate:**
 - Object ``loginAttempts[email]`` contiene: ``{count: 5, lastAttempt: timestamp, blockedUntil: timestamp+15min}``
 - Dopo scadenza blocco: entry rimossa da ``loginAttempts``
- **Esito: passed**

TC-L04: Logout e Invalidazione Sessione

- **RF Verificati:** RF25
- **Tipo Test:** Funzionale - Gestione Sessione
- **Descrizione:** Verifica che il logout rimuova il token e impedisca accessi successivi
- **Precondizioni:** Utente autenticato con sessione attiva
- **Valori di Input:** Click su "Logout" in SettingsFragment
- **Output Attesi:**
 - Token rimosso da SharedPreferences
 - Navigazione a LoginActivity
 - Back button non riporta a MainActivity
 - Chiamate API successive restituiscono 401 Unauthorized
- **Criteri di Successo:** Sessione completamente invalidata lato client
- **Risultati:**
 - **Output riscontrati:**
 - ``AuthManager.logout()`` chiamato → ``TokenManager.clearToken()`` rimuove token da SharedPreferences
 - Intent a LoginActivity con flags ``FLAG_ACTIVITY_NEW_TASK | FLAG_ACTIVITY_CLEAR_TASK``
 - Back stack completamente svuotato (`finish()` su activity corrente)
 - Chiamate successive senza token: HTTP 401 ``{error: "Token mancante"}`` da ``authenticateToken`` middleware
- **Postcondizioni riscontrate:**
 - SharedPreferences "EcoAppPrefs" non contiene più chiave "auth_token"
 - ``AuthManager.isAuthenticated()`` ritorna ``false``
 - Navigazione riparte sempre da LoginActivity
- **Esito: passed**



TC-L5: Test Non Funzionale – Performance Autenticazione

- **RF Verificati:** RNF (performance)
- **Tipo Test:** Non Funzionale - Performance
- **Descrizione:** Verifica tempi di risposta del sistema di autenticazione
- **Precondizioni:** Server attivo su Render.com, connessione stabile
- **Metriche da Verificare:**
 - | Operazione | Tempo Massimo | Note |
 - | Login (cold start) | < 5s | Render.com può avere cold start |
 - | Login (warm) | < 2s | Server già attivo |
 - | Validazione JWT | < 100ms | Operazione locale + verifica server |
- **Criteri di Successo:** 95% delle richieste entro i limiti
- **Risultati:**
 - **Output riscontrati:**
 - | Operazione | Tempo Misurato | Esito |
 - | Login cold start | ~3-4s (Render free tier spin-up) | Entro limite |
 - | Login warm | ~500ms-1s | Entro limite |
 - | Validazione JWT locale | < 10ms | Entro limite |
 - | Chiamata /api/user/profile | ~200-400ms (warm) | Accettabile |
- **Postcondizioni riscontrate:** Performance accettabile per MVP. Cold start di Render può impattare prima richiesta dopo inattività (free tier)
- **Esito:** passed

TC-L06: Opzione "Ricordami"

- **RF Verificati:** RF22
- **Tipo Test:** Funzionale - Gestione Sessione
- **Descrizione:** Verifica che la checkbox "Ricordami" estenda la durata del token
- **Precondizioni:** Utente registrato, schermata login visibile
- **Valori di Input (2 scenari):**
 - | Scenario | Checkbox | Durata Token Attesa |
 - | Senza "Ricordami" | ☐ | 7 giorni |
 - | Con "Ricordami" | ☒ | 30 giorni |
- **Output Attesi:** Token JWT con campo `exp` corrispondente alla durata
- **Verifica:** Decodifica JWT e controllo campo `exp`
- **Criteri di Successo:** Scadenza token corretta in entrambi gli scenari
- **Risultati:**
 - **Output riscontrati:**
 - Senza "Ricordami": `jwt.sign(..., { expiresIn: '7d' })` - server.js:251 `tokenExpiry = rememberMe ? '30d' : '7d'`



- Con "Ricordami": `jwt.sign(..., { expiresIn: '30d' })`
- Client invia `LoginRequest(email, password, rememberMe)` con flag booleano
- JWT payload contiene: `{id, email, iat: timestamp, exp: timestamp+durata}`
- **Postcondizioni riscontrate:**
 - Token decodificato mostra `exp` corretto (7d = +604800s, 30d = +2592000s da `iat`)
 - Checkbox `checkBoxRememberMe` in `activity_login.xml` funziona correttamente
- **Esito:** passed

TC-L07: Test Non Funzionale – Usabilità e Sicurezza Credenziali

- **RF Verificati:** RF16, RNF8, RNF11, RNF12
- **Tipo Test:** Non Funzionale - Usabilità e Sicurezza
- **Descrizione:** Verifica dell'usabilità della schermata di login e della corretta gestione delle credenziali dal punto di vista dell'utente.
- **Precondizioni:**
 1. Utente reale che non ha mai utilizzato l'app.
 2. Strumento di osservazione/test di usabilità disponibile.
- **Valori di Input:**
 1. Task: "Apri l'app, effettua il login e poi disconnettiti".
 2. Osservazione dei passaggi dell'utente (senza assistenza).
- **Output Attesi:**
 1. L'utente deve comprendere immediatamente dove inserire email e password e come confermare l'accesso (coerenza UI, RNF8).
 2. I messaggi di errore visualizzati, in caso di inserimento errato, devono essere chiari ma non rivelare quale campo è specificamente errato (RNF12).
 3. Durante il test non devono emergere evidenze di memorizzazione in chiaro delle credenziali (es. password visibile in log, screenshot, ecc.), coerentemente con RNF11.
- **Criteri di Successo:** UI intuitiva, messaggi generici, nessuna esposizione password
- **Risultati:**
 1. **Output riscontrati:**
 - Usabilità UI (RNF8):
 - `activity_login.xml` usa Material Design con `TextInputLayout` + hint chiari ("Email", "Password")
 - Pulsante "ACCEDI" prominente e ben visibile
 - Link "Non hai un account? Registrati" chiaro per navigazione
 - Layout responsive con `ConstraintLayout`
 2. **Messaggi errore generici (RNF12):**
 - Server risponde sempre `{error: "Credenziali non valide"}` sia per email inesistente che password errata (server.js:236, 241)
 - NON viene rivelato se l'email esiste o meno nel sistema
 - Rate limiting mostra: "Troppi tentativi falliti" (nessun dettaglio su quale campo)
 3. **Sicurezza credenziali (RNF11):**



- - Password MAI loggata in chiaro (nessun `console.log(password)` nel codebase)
- - Password hashata con `bcrypt.hash(password, 10)` prima del salvataggio
- - Campo password con `inputType="textPassword"` (mascherato con `•••`)
- - Token JWT NON contiene password (solo `{id, email, iat, exp}`)
- - SharedPreferences salva solo token, MAI credenziali
- **Postcondizioni riscontrate:**
 1. Flusso login completabile in < 30 secondi da utente non esperto
 2. Nessuna informazione sensibile esposta in log, UI o storage
 3. Messaggi di errore non permettono enumeration attack (non rivelano esistenza email)
- **Esito:** passed

Registrazione

I seguenti casi di test sono progettati per validare i requisiti funzionali **RF27–RF38** e i requisiti non funzionali **RNF13–RNF17** relativi al caso d'uso "Creazione Utente / Registrazione (gestione amministrativa)".

TC-R01: Accesso Schermata Registrazione

- **RF Verificati:** RF27
- **Tipo Test:** Funzionale - Navigazione
- **Descrizione:** Verifica che un visitatore non autenticato possa accedere alla schermata di registrazione tramite il link nella schermata login
- **Precondizioni:**
 1. App avviata
 2. Nessun token salvato (utente non autenticato)
 3. LoginActivity visibile
- **Valori di Input:** Click su link/pulsante "Registrati" o "Non hai un account?"
- **Output Attesi:**
 1. Navigazione a RegisterActivity
 2. Form visibile con campi:
 - Nome (EditText, obbligatorio)
 - Email (EditText, obbligatorio, validazione formato)
 - Password (EditText, obbligatorio, min 6 caratteri)
 - Conferma Password (EditText, obbligatorio)
 3. Pulsante "Registrati" visibile
 4. Link "Hai già un account? Accedi" visibile
- **Criteri di Successo:** Schermata registrazione accessibile e completa
- **Risultati:**
 1. **Output riscontrati:**
 - Click su `textViewRegisterLink` - `navigateToRegister()` - `Intent(this, RegisterActivity.class)`



- `activity_register.xml` contiene:
 - `editTextName` (TextInputEditText)
 - `editTextEmail` (TextInputEditText)
 - `editTextPassword` (TextInputEditText, inputType="textPassword")
 - `editTextConfirmPassword` (TextInputEditText, inputType="textPassword")
- `buttonRegister` presente e funzionante
- `textViewLoginLink` - `finish()` torna a LoginActivity
- **Postcondizioni riscontrate:** RegisterActivity visibile, form vuoto e pronto per input, keyboard focus su primo campo
- - **Esito:** passed

TC-R02: Registrazione Self-Service Completa

- **RF Verificati:** RF28, RF29, RF30, RF31, RF32, RF33, RF34
- **Tipo Test:** Funzionale - Scenario Nominale (End-to-End)
- **Descrizione:** Verifica del flusso completo di auto-registrazione da parte di un visitatore
- **Precondizioni:**
 1. Email "nuovo.utente@example.com" NON presente nel database
 2. Connessione internet attiva
 3. RegisterActivity visibile
- **Valori di Input:**
 1. Nome: "Mario Rossi"
 2. Email: "nuovo.utente@example.com"
 3. Password: "SecurePass123"
 4. Conferma Password: "SecurePass123"
- **Passi Esecuzione:**
 1. Compilare tutti i campi
 2. Click su "Registrati"
 3. Attendere risposta server
- **Output Attesi:**
 1. Richiesta POST a `/api/auth/register` inviata
 2. Risposta HTTP 201 Created
 3. Documento utente creato in MongoDB con:
 - `name`: "Mario Rossi"
 - `email`: "nuovo.utente@example.com"
 - `password`: hash bcrypt (NON in chiaro)
 - `totalPoints`: 0
 - `co2Saved`: 0
 - `level`: "Eco-Novizio"
 - `badges`: [badge iniziale]
 - `createdAt`: timestamp
 4. Email di benvenuto inviata (tramite Resend)
 5. Toast "Registrazione completata!" visibile



6. Navigazione automatica a LoginActivity

- **Criteri di Successo:** Utente creato, email inviata, redirect corretto

- **Risultati:**

1. **Output riscontrati:**

- POST `/api/auth/register` con body: `{email, password, name}` (RegisterRequest.java)
- Server risponde HTTP 201 con JSON:

```
``json
{
  "message": "Registrazione completata",
  "token": "eyJ...",
  "user": {id, email, name, nickname, level: "Eco-Novizio", totalPoints: 0,
co2Saved: 0, badges: [{id:1, name:"Eco-Novizio"...}]}
}
...`
```
- Password hashata con `bcrypt.hash(password, 10)` - mai salvata in chiaro
- `sendWelcomeEmail(user.email, user.name)` chiamato (se RESEND_API_KEY configurata)
- Toast: "Benvenuto {name}!" + animazione confetti (Konfetti library)
- Dopo 2s delay: `navigateToMain()` - MainActivity (NON LoginActivity - login automatico post-registrazione)

- **Postcondizioni riscontrate:**

1. Utente creato in MongoDB con `_id` generato
2. UserQuest inizializzate per tutte le GlobalQuest esistenti
3. Token JWT salvato - utente già autenticato
4. Email benvenuto inviata (o errore loggato silenziosamente)

- - **Esito:** passed

TC-R03: Gestione Email Duplicata

- **RF Verificati:** RF31, RF35
- **Tipo Test:** Funzionale - Scenario Alternativo (Errore)
- **Descrizione:** Verifica che il sistema blocchi la registrazione con email già esistente
- **Precondizioni:**
 1. Email "esistente@example.com" GIÀ presente nel database
 2. RegisterActivity visibile
- **Valori di Input:**
 1. Nome: "Utente Duplicato"
 2. Email: "esistente@example.com" // già registrata
 3. Password: "Password123"
 4. Conferma Password: "Password123"
- **Output Attesi:**
 1. Richiesta POST a `/api/auth/register` inviata



2. Risposta HTTP 400 Bad Request
 3. Messaggio errore: "Email già registrata" o simile
 4. Form NON resettato (dati preservati per correzione)
 5. Nessun documento creato nel database
 6. Nessuna email inviata
- **Criteri di Successo:** Registrazione bloccata, messaggio chiaro, dati preservati
 - **Risultati:**
 1. **Output riscontrati:**
 - POST `/api/auth/register` inviato normalmente
 - Server verifica: `User.findOne({ email: email.toLowerCase() })` - utente trovato
 - HTTP 409 Conflict (non 400) con `{error: "Email già registrata"}` - server.js:172
 - Client riceve errore in `onResponse()` con `response.code() != 2xx`
 - Toast mostra: "Email già registrata" (estratto da ErrorResponse)
 - Form preservato - nessun reset dei campi, utente può correggere email
 - **Postcondizioni riscontrate:**
 1. Nessun nuovo documento in MongoDB
 2. `sendWelcomeEmail()` non chiamato
 3. RegisterActivity rimane visibile con dati inseriti
 4. Esito: PASSED
 5. Nota: HTTP status è 409 (Conflict)

TC-R04: Resilienza Invio Email

- **RF Verificati:** RF33, RF38
- **Tipo Test:** Funzionale - Gestione Errori
- **Descrizione:** Verifica che la registrazione venga completata anche se l'invio email fallisce
- **Precondizioni:**
 1. Email valida e non esistente
 2. Simulazione: servizio Resend non raggiungibile o errore API
 3. Valori di Input: Dati registrazione validi
- **Output Attesi:**
 1. Utente creato con successo nel database
 2. Account IMMEDIATAMENTE ATTIVO
 3. Errore email loggato nel server (console.error)
 4. Toast "Registrazione completata!" comunque mostrato
 5. Redirect a LoginActivity
 6. Utente può effettuare login immediatamente
- **Criteri di Successo:** Registrazione NON fallisce per errore email
- **Risultati:**
 1. **Output riscontrati:**
 - `sendWelcomeEmail()` chiamato DOPO `user.save()` e `res.status(201).json(...)` - server.js:201-203
 - Email inviata in modo asincrono (no `await` prima della response)



- Se RESEND_API_KEY non configurata: ``console.log('Email non inviata (Resend non configurato)')`` - nessun errore
- Se Resend fallisce: ``console.error('Email error:', error.message)`` - catch silenzioso
- Response HTTP 201 già inviata PRIMA del tentativo email
- Client riceve successo e procede con confetti + navigazione
- **Postcondizioni riscontrate:**
 1. Utente creato e attivo indipendentemente dall'esito email
 2. Token JWT generato e restituito
 3. Errore email visibile solo nei log del server, mai propagato al client
- **Esito:** passed

TC-R05: Validazione Form Registrazione

- **RF Verificati:** RF30, RNF13
- **Tipo Test:** Funzionale - Validazione Input
- **Descrizione:** Verifica validazione client-side del form di registrazione
- **Precondizioni:** RegisterActivity visibile
- **Valori di Input (scenari multipli):**
 1. | # | Nome | Email | Password | Conferma | Risultato Atteso |
 2. | 1 | "" | "a@b.c" | "pass123" | "pass123" | Errore "Nome obbligatorio" |
 3. | 2 | "Mario" | "" | "pass123" | "pass123" | Errore "Email obbligatoria" |
 4. | 3 | "Mario" | "invalid" | "pass123" | "pass123" | Errore "Email non valida" |
 5. | 4 | "Mario" | "a@b.c" | "12345" | "12345" | Errore "Password min 6 caratteri" |
 6. | 5 | "Mario" | "a@b.c" | "pass123" | "pass456" | Errore "Le password non coincidono" |
 7. | 6 | "Mario" | "a@b.c" | "pass123" | "pass123" | Form valido, pulsante abilitato |
- **Criteri di Successo:** Tutti gli scenari producono il risultato atteso
- **Risultati:**
 1. **Output riscontrati:**
 - | # | Input | Output Effettivo | Componente |
 - | 1 | Nome vuoto | ``textInputLayoutName.setError("Nome richiesto")`` | RegisterActivity.java:118 |
 - | 1b | Nome < 2 char | ``textInputLayoutName.setError("Nome troppo corto")`` | RegisterActivity.java:122 |
 - | 2 | Email vuota | ``textInputLayoutEmail.setError("Email richiesta")`` | RegisterActivity.java:128 |
 - | 3 | Email invalida | ``textInputLayoutEmail.setError("Email non valida")`` | RegisterActivity.java:132 (usa ``Patterns.EMAIL_ADDRESS``) |
 - | 4 | Password < 6 | ``textInputLayoutPassword.setError("Password troppo corta (min 6 caratteri)")`` | RegisterActivity.java:142 |
 - | 5 | Password diversa da Conferma | ``textInputLayoutConfirmPassword.setError("Le password non corrispondono")`` | RegisterActivity.java:152 |
 - | 6 | Tutto valido | ``validateInput()`` ritorna ``true``, chiamata API procede | - |



- **Postcondizioni riscontrate:**
 1. Errori visualizzati inline sotto ciascun campo con ``TextInputLayout.setError()``
 2. Validazione sequenziale (primo errore trovato blocca)
 3. Errori resettati con ``setError(null)`` quando campo corretto
- **Esito:** passed

TC-R06: Tracciabilità Creazione Account

- **RF Verificati:** RF37
- **Tipo Test:** Non Funzionale - Tracciabilità
- **Descrizione:** Verifica che il sistema registri automaticamente la data/ora di creazione account
- **Precondizioni:** Registrazione completata con successo
- **Verifica:**
 1. Query MongoDB: ``db.users.findOne({email: "nuovo@test.it"})``
 2. Verificare presenza campo ``createdAt``
 3. Verificare che ``createdAt`` sia timestamp valido (ISO 8601)

- **Output Attesi:**

```
```json
{
 "_id": ObjectId("..."),
 "name": "Nuovo Utente",
 "email": "nuovo@test.it",
 "createdAt": "2026-01-25T10:30:00.000Z", // <- verificare
 "updatedAt": "2026-01-25T10:30:00.000Z"
}
```
```

- **Criteri di Successo:** Campo ``createdAt`` presente e valorizzato
- **Risultati:**

1. **Output riscontrati:**

- Schema User in ``models/User.js`` usa ``{ timestamps: true }`` opzione Mongoose
- Documento creato contiene automaticamente:

```
```json
{
 "_id": ObjectId("..."),
 "name": "Test User",
 "email": "test@example.com",
 "password": "$2a$10$...", // bcrypt hash
 "totalPoints": 0,
 "co2Saved": 0,
 "level": "Eco-Novizio",
 "badges": [{id: 1, name: "Eco-Novizio", ...}],
 "createdAt": ISODate("2026-01-25T10:30:00.000Z"),
 "updatedAt": ISODate("2026-01-25T10:30:00.000Z"),
}
```



```
"__v": 0
}
```

- 3. `createdAt` e `updatedAt` gestiti automaticamente da Mongoose

- **Postcondizioni riscontrate:**

1. Ogni utente ha timestamp di creazione persistente
2. Campo `updatedAt` aggiornato automaticamente ad ogni modifica
3. Nessun campo "createdBy" o audit log (come da architettura self-service)

- **Esito:** passed

## 4. Schedule del testing

### Profilo

Test	Nome	Priorità	Durata stimata	Risorse
TC-P01	Visualizzazione profilo	Alta	2 min	1 tester, dispositivo Android, DB popolato
TC-P02	Errore di rete	Alta	10 min	1 tester, dispositivo con modalità aereo
TC-P04	Performance caricamento	Alta	10 min	1 tester, cronometro, 4G stabile
TC-P05	Sicurezza accesso indebito	Alta	20-30 min	1 tester, account multipli, strumenti API (Postman)
TC-P06	Usabilità interfaccia	Media	10 min	1 utente esterno, 1 osservatore
TC-P07	Compatibilità	Media	10 min	Emulatore/dispositiv



Android 8

ivo Android 8

### Quest

Test	Nome	Priorità	Durata stimata	Risorse
TC-Q01	Assegnazione quest	Alta	10-15 min	1 tester, dispositivo Android, quest disponibili
TC-Q02	Completamento quest	Alta	10-15 min	1 tester, rete attiva, DB con quest assegnate
TC-Q03	Sistema punti	Alta	5-10 min	1 tester, quest attive, accesso DB
TC-Q04	Quest ripetibili disponibili	Alta	10 min	1 tester, almeno una quest ripetibile
TC-Q05	Contatore progressi	Media	10-20 min	1 tester, quest ripetibile con contatore
TC-Q07	Usabilità lista quest	Media	20-40 min	1 utente esterno + 1 osservatore
TC-Q08	Storico e statistiche	Bassa/Media	10-15 min	1 tester, storico quest popolato

### Login

Test	Nome	Priorità	Durata stimata	Risorse
TC-L01	Login corretto	Alta	10-15 min	1 tester, utente attivo nel DB, app e connessione
TC-L02	Validazione campi email/password	Alta	10-15 min	1 tester, app aperta sul login
TC-L03	Blocco account tentativi falliti	Alta	20-30 min	1 tester, account con tentativi configurati, password errate
TC-L04	Logout e invalidazione	Alta	10 min	1 tester, utente autenticato, app



sessione				con sezioni riservate
TC-L05	Performance autenticazione	Alta	10 min	1 tester, cronometro, rete 4G/Wi-Fi, account attivo
TC-L07	Usabilità e sicurezza credenziali	Media	20-40 min	1 utente esterno + osservatore, app aperta sul login
TC-L06	Funzionalità Ricordami	Media	15-20 min	1 tester, account attivo, verificatoken 30gg

### Registrazione

Test	Nome	Priorità	Durata Stimata	Risorse Necessarie
TC-R01	Accesso alla schermata registrazione	Alta	5-10 min	1 tester, app su loginactivity
TC-R02	Registrazione self service completa	Alta	10-15 min	1 tester, DB funzionante, email non esistente
TC-R03	Gestione email duplicata	Alta	15-20 min	1 tester, DB con utente esistente
TC-R05	Validazione form registrazione	media	10-15 min	1 tester, scenari input invalidi
TC-R06	Tracciabilità creazione account	Media	15-20 min	1 tester, accesso mongodb per verifica timestamp
TC-R04	Resilienza invio mail	Media	10-20 min	1 tester, resend_api_key disabilitata

## 5. Procedure di registrazione dei test

La procedura di registrazione dei test definisce come devono essere documentati gli esiti dei singoli casi di test, garantendo **tracciabilità**, **ripetibilità** e **coerenza** con le linee guida presentate nel materiale didattico.

In conformità alla slide “Caso di test – Riferimento a un requisito”, ogni caso di test è descritto da una scheda che contiene **almeno i seguenti campi**:



- Numero identificativo (TC-XXX)
- Descrizione funzionalità e RF verificati
- Precondizioni e valori di input
- Output attesi e criteri di successo

**Approccio di testing adottato:**

Testing manuale esplorativo con verifica white-box del codice sorgente e approccio black box per aggiungere eventuali bug rilevati da experience.

L'esecuzione dei test è stata integrata nel ciclo di sviluppo iterativo:

- Implementazione funzionalità
- Test manuale (app + API)
- Identificazione bug
- Fix e commit con messaggio descrittivo
- Re-test di verifica

**Registrazione esiti:**

- Output riscontrati (con riferimenti a codice sorgente)
- Postcondizioni riscontrate
- Esito (PASSED/FAILED)

**Tracciabilità:**

- Git history come audit trail (70 commit, di cui ~12 relativi a fix/bug)
- Commit messages documentano correzioni (es. "Fix: quest non si attivavano")

**Strumenti utilizzati:**

- Android Studio (emulatore, logcat, debugger)
- Postman/curl (test API REST)
- MongoDB Compass (verifica stato database)
- Browser DevTools (debug network)

**Archiviazione:**

- Test cases nel Piano di Testing (D4)
- Risultati integrati nei test cases (sezione "Risultati")
- Storicità garantita da versioning Git
- Bug significativi tracciabili via GitHub Issues

## 6. Requisiti hardware e software

I requisiti hardware e software definiscono l'ambiente minimo e raccomandato per garantire una corretta esecuzione del testing, in coerenza con i requisiti non funzionali (interoperabilità, robustezza, prestazioni).



## Hardware minimo

### Smartphone Android (per test applicazione utente)

- Android 8.0 (Oreo) o superiore
- CPU quad-core
- RAM  $\geq 2$  GB
- Spazio libero  $\geq 500$  MB

### PC/Laptop (sviluppo e test API)

- CPU dual-core
- RAM  $\geq 8$  GB
- Browser aggiornato (Chrome, Firefox, Edge)

## Hardware consigliato

- Smartphone Android con:
  - Android 10+
  - RAM  $\geq 4$  GB
  - Storage  $\geq 1$  GB
- Emulatore Android (Android Studio) per test ripetibili, cattura logcat e simulazioni.
- PC con:
  - CPU i5 o equivalente
  - RAM  $\geq 16$  GB
  - SSD

## Software necessario

- App Android:
  - Ultima build APK dell'app
  - Android Studio (emulatore + logcat + debugger)
- Backend:
  - Node.js v18+ (runtime server)
  - npm o yarn (gestione dipendenze)
- Database
  - MongoDB Compass (verifica dati, query manuali)
  - mongoDB Atlas (db cloud in produzione)
- Testing API:
  - Postman / Thunder Client / curl
  - Token JWT valido per endpoint autenticati
- Versionamento e Tracking:
  - Git + GitHub (versioning)
  - GitHub Issues (bug tracking)
- Hosting
  - [render.com](https://render.com)



## Ambiente di rete

- Connessione Wi-Fi stabile
- Possibilità di testare anche in rete 4G
- Possibilità di simulare condizioni di rete degradate o assenti, in accordo con i test su robustezza e tolleranza agli errori (slide “Requisiti Non Funzionali”)

Per garantire la verifica della **interoperabilità**, i test verranno eseguiti – quando possibile – su **più versioni di Android e differenti risoluzioni dello schermo**, utilizzando sia dispositivi fisici che emulatori.

## 7. Vincoli che condizionano il testing

I vincoli rappresentano le limitazioni tecniche, organizzative e operative che influenzano la pianificazione e l'esecuzione del testing. La loro identificazione è coerente con i temi di risorse disponibili, complessità dei test, e qualità attesa del sistema.

### Vincoli tecnici

1. **Compatibilità minima Android (RNF26)**  
Alcune funzionalità non possono essere testate su versioni precedenti ad Android 8.
2. **Testing ibrido con prevalenza tipo Black-box**  
Non si ha (volutamente) accesso al codice sorgente; la verifica avviene esclusivamente attraverso comportamenti osservabili.
3. **Dipendenza da servizi esterni**
  - Backend su Render.com (cold start 30-60s su free tier dopo inattività)
  - MongoDB Atlas (cluster condiviso, limiti di connessioni)
  - Resend API (email benvenuto - non bloccante, fallimento silenzioso)
4. **Limiti nella simulazione di condizioni estreme**  
Stress test intensivi o test di carico elevato non possono essere effettuati, in linea con quanto discusso nelle slide su *prestazioni e scalabilità*.
5. **Variabilità delle performance di rete**  
I requisiti di performance (es. RNF9) sono influenzati dalla qualità della rete. Eventuali anomalie devono essere validate controllando la stabilità della connessione.

### Vincoli organizzativi



1. **Ruoli e permessi**

I test di registrazione possono essere eseguiti da qualsiasi visitatore non autenticato

2. **Disponibilità del team**

Il testing deve rispettare la pianificazione dello schedule (sezione 4).

3. **Scadenze del progetto formativo**

Le fasi di test devono essere completate entro le consegne programmate delle esercitazioni.

## Vincoli operativi

1. **Numero limitato di dispositivi fisici**

È possibile che alcuni test vengano eseguiti solo su emulatori.

2. **Impossibilità di riprodurre tutte le condizioni reali**

Test come la simulazione di *affidabilità estesa nel tempo* o *condizioni estreme d'uso* non possono essere completati integralmente.

3. **Dati e account fittizi**

Per ragioni di privacy tutti i test devono utilizzare dati di prova, non reali.

4. **Ambienti non identici all'ambiente di produzione**

Eventuali differenze tra ambienti potrebbero causare scostamenti nelle performance (coerente con slide sulla qualità del sistema e prestazioni).