

# 1. ANALISI DEI REQUISITI

---

## 1.1. DECOMPOSIZIONE DEL TESTO

### UTENTE

- Tutti gli utenti della piattaforma dispongono di: **email** (univoca), **nickname**, **password**, **nome**, **cognome**, **anno di nascita**, e un **luogo di nascita**. Inoltre, ogni utente può indicare le proprie skill di curriculum
- Gli utenti **possono appartenere** (non necessariamente) a due sotto-categorie: **amministratori** e **creatori**
- **Ogni utente** della piattaforma può **finanziare un progetto**
- Un utente può **candidarsi ad un numero qualsiasi di profili**

### SKILL DI CURRICULUM

- Le skill di curriculum consistono in una sequenza di: **<competenza, livello>**, dove **la competenza è una stringa** ed il **livello è un numero tra 0 e 5** (es. <AI, 3>)
- La lista delle competenze è **comune a tutti gli utenti** della piattaforma

### AMMINISTRATORE

- Gli utenti amministratori dispongono anche di **un codice di sicurezza**
- **Solo** gli utenti **amministratori** possono **popolare la lista delle competenze**

### CREATORE

- Gli utenti creatori dispongono anche dei campi: **nr\_progetti** ed **affidabilità**
- **Solo** un utente **creatore** può **inserire uno o più progetti**
- L'utente creatore può eventualmente **inserire una risposta per ogni singolo commento** (un commento ha al **massimo 1 risposta**)
- L'utente creatore può **accettare o meno la candidatura** di un potenziale partecipante del **proprio progetto software**

### PROGETTO GENERICO

- Ogni progetto dispone di: un **nome** (univoco), un **campo descrizione**, una **data di inserimento**, **una o più foto**, un **budget** da raggiungere per avviare il progetto, una **data limite** entro cui raggiungere il budget, uno **stato**. Lo stato è un campo di tipo **enum (aperto/chiuso)**
- Ogni progetto è **associato ad un solo utente creatore**
- Ogni progetto prevede una **lista di reward**
- Ogni progetto appartiene esclusivamente ad una di due categorie: progetti **hardware** o progetti **software**
- Nel momento in cui la **somma totale degli importi dei finanziamenti supera il budget** del progetto, oppure il progetto **resta in stato aperto oltre la data limite**, lo stato di tale progetto **diventa pari a chiuso**
- Un **progetto chiuso** non accetta **ulteriori finanziamenti**

## REWARD

- Una reward dispone di: un **codice** (univoco), una **breve descrizione**, una **foto**

## PROGETTO HARDWARE

- Nel caso dei progetti hardware, è presente anche la **lista delle componenti necessarie**

## COMPONENTE HARDWARE

- Ogni componente ha: un **nome** (univoco), una **descrizione**, un **prezzo**, una **quantità (>0)**

## PROGETTO SOFTWARE

- Nel caso dei progetti software, viene elencata la **lista dei profili necessari** per lo sviluppo
- Un progetto software **può ricevere un numero qualsiasi di candidature per un certo profilo**

## PROFILO

- Ogni profilo dispone di: un **nome** (es. “Esperto AI”) e di **skill richieste**

## SKILL DI PROFILO

- Le **skill di profilo** consistono in una sequenza <competenza, livello>, dove la **competenza** è una **stringa** (tra quelle **presenti in piattaforma**) ed il **livello** è un **numero tra 0 e 5**

## FINANZIAMENTO

- Ogni finanziamento dispone di: un **importo** ed una **data**.
- Un utente potrebbe inserire **più finanziamenti per lo stesso progetto**, ma in **date diverse**
- Ad ogni finanziamento è associata **una sola reward**, tra quelle **previste per il progetto finanziato**

## COMMENTO

- Un utente può **inserire commenti relativi ad un progetto**. Ogni commento dispone di: un **id** (univoco), una **data** ed un campo **testo**

## PARTECIPANTE

- È prevista la possibilità per gli **utenti** di **candidarsi come partecipanti allo sviluppo di un progetto software**
- La piattaforma consente ad un utente di inserire una candidatura su un profilo **SOLO se, per ogni skill richiesta da un profilo, l'utente dispone di un livello superiore o uguale** al valore richiesto

## LOG (MongoDB)

- Si vuole tenere traccia di **tutti gli eventi che occorrono nella piattaforma**, relativamente **all'inserimento di nuovi dati** (es. nuovi utenti, nuovi progetti, etc)
- Tali eventi vanno inseriti, sotto forma di **messaggi di testo**, **all'interno di un log**, implementato in un' apposita **collezione MongoDB**

## 1.2. LISTA DELLE OPERAZIONI

Operazioni che riguardano **TUTTI** gli utenti:

- **Autenticazione/Gestione Account**
  - Login/logout dalla piattaforma
  - Registrazione con ruoli opzionali (creatore, admin)

- **Curriculum**
  - Inserimento, aggiornamento e rimozione delle proprie competenze
  - Visualizzazione del proprio curriculum
- **Progetti**
  - Visualizzazione dei progetti disponibili
  - Visualizzazione delle statistiche della piattaforma
- **Finanziamenti**
  - Effettuare finanziamenti su progetti aperti
  - Selezione di reward associate ai finanziamenti
  - Visualizzazione dei propri finanziamenti
- **Interazioni**
  - Inserimento e cancellazione dei propri commenti
  - Invio di candidature per profili di progetti software
  - Visualizzazione dello stato delle proprie candidature

Operazioni che riguardano SOLO gli amministratori:

- Gestione delle competenze globali (inserimento, aggiornamento)
- Autenticazione con codice di sicurezza aggiuntivo
- Cancellazione dei commenti di qualsiasi utente
- Visualizzazione del registro delle attività (logs)

Operazioni che riguardano SOLO i creatori:

- **Gestione Progetti**
  - Creazione di nuovi progetti (software/hardware)
  - Aggiornamento descrizione e budget dei progetti
  - Gestione delle foto dei progetti (caricamento/eliminazione)
- **Gestione Componenti** (per progetti hardware)
  - Inserimento, aggiornamento e rimozione di componenti
- **Gestione Profili** (per progetti software)
  - Creazione, modifica e cancellazione di profili
  - Gestione delle competenze richieste per ciascun profilo
- **Gestione Candidature**
  - Accettazione o rifiuto delle candidature
- **Gestione Reward**
  - Creazione di reward per i progetti
- **Interazioni**
  - Inserimento e cancellazione di risposte ai commenti

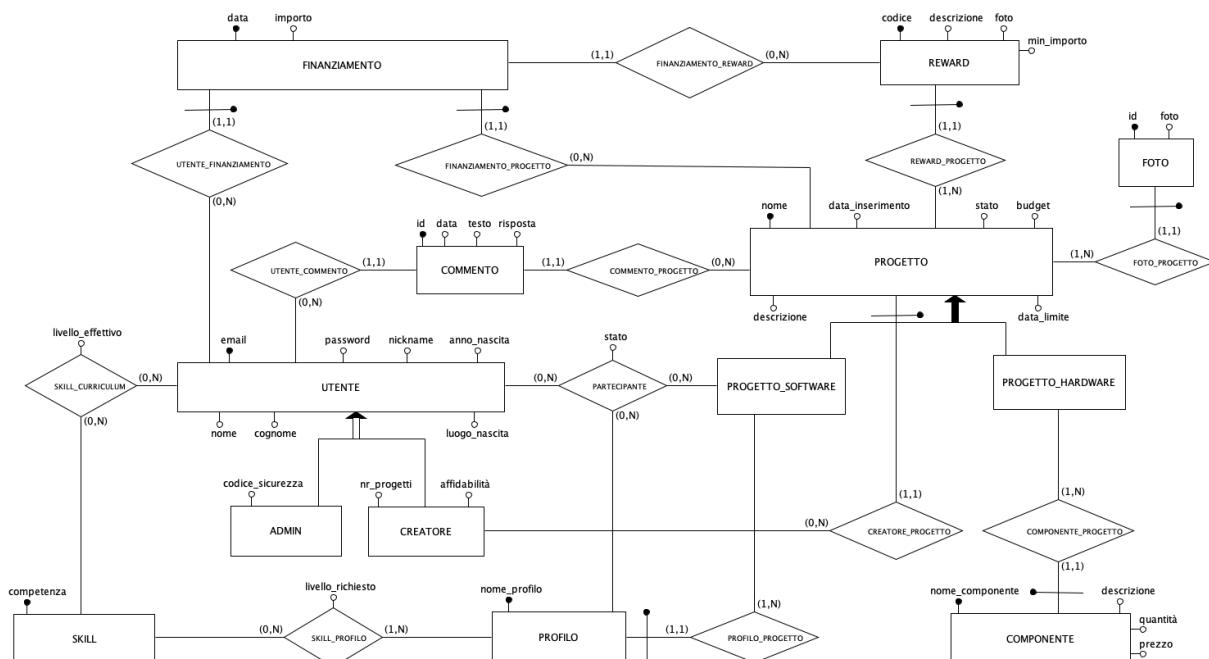
# 1.3. GLOSSARIO DEI DATI

TERMINE	DESCRIZIONE	SINONIMI	COLLEGAMENTI
<b>UTENTE</b>	Un utente generico della piattaforma BOSTARTER. Generalizzazione (non totale) di AMMINISTRATORE/CREATORE.		AMMINISTRATORE / CREATORE, SKILL DI CURRICULUM, FINANZIAMENTO, PARTECIPANTE, COMMENTO
<b>SKILL DI CURRICULUM</b>	Le competenze specifiche di ogni utente.		UTENTE
<b>AMMINISTRATORE</b>	Un utente privilegiato della piattaforma, può modificare dati sensiti come la lista delle competenze. Essendo una specializzazione di UTENTE, è associato in via indiretta ad ogni collegamento di quest'ultimo.		UTENTE
<b>CREATORE</b>	Un utente che ha il permesso di aprire e gestire progetti (propri) sulla piattaforma. Essendo una specializzazione di UTENTE, è associato in via indiretta ad ogni collegamento di quest'ultimo.		UTENTE, PROGETTO GENERICO / HARDWARE / SOFTWARE
<b>PROGETTO GENERICO</b>	Un progetto generico sulla piattaforma; si tratta di una generalizzazione (totale) di PROGETTO HARDWARE/SOFTWARE. Viene gestito da un solo utente creatore.	Progetto	PROGETTO HARDWARE / SOFTWARE, CREATORE, REWARD, FINANZIAMENTO, COMMENTO
<b>REWARD</b>	Un premio associato ad un progetto, viene offerto ad utenti che finanziato il progetto.	Premio	PROGETTO GENERICO, FINANZIAMENTO
<b>PROGETTO HARDWARE</b>	Un progetto specifico all'ambito hardware, dispone di una lista di componenti necessarie. Specializzazione di PROGETTO GENERICO.		PROGETTO GENERICO, CREATORE, COMPONENTE HARDWARE
<b>PROGETTO SOFTWARE</b>	Un progetto specifico nell'ambito software, dispone di una lista di profili richiesti per lo sviluppo, e può avere più partecipanti. Specializzazione di PROGETTO GENERICO.		PROGETTO GENERICO, CREATORE, PARTECIPANTE, PROFILO
<b>PROFILO</b>	Un profilo particolare richiesto per un progetto software, dispone di un nome ed una lista di skill richieste. Potenziali partecipanti per un progetto software si misurano sulla		PROGETTO SOFTWARE, SKILL DI PROFILO

TERMINI	DESCRIZIONE	SINONIMI	COLLEGAMENTI
	base del profilo e livello delle skill richieste.		
<b>SKILL DI PROFILO</b>	Una competenza specifica appartenente ad un profilo per progetti software.		PROFILO
<b>FINANZIAMENTO</b>	Un finanziamento economico fatto da un qualunque tipo di utente per un progetto hardware/software.		UTENTE, PROGETTO GENERICO, REWARD
<b>COMMENTO</b>	Un commento fatto da un qualunque tipo di utente per un progetto hardware/software. Può contenere al massimo una risposta da parte dell'utente creatore.		UTENTE, PROGETTO GENERICO
<b>PARTECIPANTE</b>	Un potenziale o effettivo partecipante ad un progetto software. Qualunque utente che non è creatore del progetto software può candidarsi ad esso se dispone delle skill e livelli necessari, e può essere accettato/rifiutato dall'utente creatore.	Candidato	UTENTE, PROGETTO SOFTWARE

## 2. PROGETTAZIONE CONCETTUALE

### 2.1. DIAGRAMMA E-R



## 2.2. DIZIONARIO DELLE ENTITÀ

ENTITÀ	DESCRIZIONE	ATTRIBUTI	IDENTIFICATORE
<b>UTENTE</b>	Utente generico della piattaforma BOSTARTER. Ogni utente (admin/creatore) viene inglobato in questa entità.	email, password, nickname, nome, cognome, anno_nascita, luogo_nascita	email
<b>ADMIN</b>	Specializzazione di UTENTE. Vengono inseriti gli utenti privilegiati/amministratori.	email_utente, codice_sicurezza	email_utente
<b>CREATORE</b>	Specializzazione di UTENTE. Vengono inseriti gli utenti che creano e gestiscono progetti.	email_utente, nr_progetti, affidabilità	email_utente
<b>PROGETTO</b>	Un progetto generico della piattaforma. Ogni progetto (software/hardware) viene inglobato in questa entità.	nome, email_creatore, descrizione, budget, stato, data_inserimento, data_limite	nome
<b>FOTO</b>	Una o più foto associate ad un determinato progetto.	id, nome_progetto, foto	id, nome_progetto
<b>RWARD</b>	Una o più reward associate ad un determinato progetto.	codice, nome_progetto, descrizione, foto, min_importo	codice, nome_progetto
<b>COMMENTO</b>	Uno o più commenti associati ad un utente ed un progetto. Opzionalmente contengono una risposta dall'utente creatore.	id, email_utente, nome_progetto, data, testo, risposta	id
<b>PROGETTO_SOFTWARE</b>	Specializzazione di PROGETTO. Vengono inseriti i progetti di tipo software.	nome_progetto	nome_progetto
<b>PROGETTO_HARDWARE</b>	Specializzazione di PROGETTO. Vengono inseriti i progetti di tipo hardware.	nome_progetto	nome_progetto
<b>COMPONENTE</b>	Una o più componenti fisiche necessarie per un progetto hardware.	nome_componente, nome_progetto, descrizione, quantità, prezzo	nome_componente nome_progetto

ENTITÀ	DESCRIZIONE	ATTRIBUTI	IDENTIFICATORE
<b>PROFILO</b>	Un profilo aperto ad utenti qualificati per partecipare ad un progetto software.	nome_profilo	nome_profilo
<b>SKILL</b>	Una o più competenze richieste/disponibili sulla piattaforma per progetti software. La lista di competenze è gestita dagli amministratori.	competenza	competenza
<b>FINANZIAMENTO</b>	Un finanziamento economico fatto da un utente, verso un progetto, associato ad una reward.	data, email_utente, nome_progetto, codice_reward, importo	data, email_utente, nome_progetto

## 2.3. DIZIONARIO DELLE RELAZIONI

RELAZIONI	DESCRIZIONE	COMPONENTI	ATTRIBUTI
<b>SKILL_CURRICULUM</b>	Le competenze effettive di ciascun utente, tratte da SKILL ed associate al livello dell'utente.	UTENTE, SKILL	livello_effetti
<b>UTENTE_FINANZIAMENTO</b>	L'associazione di un finanziamento all'utente che l'ha fatto.	UTENTE, FINANZIAMENTO	
<b>UTENTE_COMMENTO</b>	L'associazione di un commento all'utente che l'ha postato.	UTENTE, COMMENTO	
<b>PARTECIPANTE</b>	Un utente e potenziale candidato ad un profilo per partecipare ad un progetto software esistente. Candidati che non possiedono le competenze/livelli necessari non vengono considerati.	UTENTE, PROGETTO_SOFTWARE, PROFILO	stato

RELAZIONI	DESCRIZIONE	COMPONENTI	ATTRIBUTI
<b>FINANZIAMENTO_PROGETTO</b>	L'associazione di un finanziamento ad un progetto.	FINANZIAMENTO, PROGETTO	
<b>COMMENTO_PROGETTO</b>	L'associazione di un commento ad un progetto.	COMMENTO, PROGETTO	
<b>REWARD_PROGETTO</b>	L'associazione di una reward ad un progetto.	REWARD, PROGETTO	
<b>CREATORE_PROGETTO</b>	L'associazione di un utente creatore al progetto che ha creato.	CREATORE, PROGETTO	
<b>FOTO_PROGETTO</b>	La lista di una o più foto che rappresentano un progetto.	FOTO, PROGETTO	
<b>FINANZIAMENTO_REWARD</b>	L'associazione di una reward per un finanziamento di un progetto.	FINANZIAMENTO, REWARD	
<b>COMPONENTE_PROGETTO</b>	La lista di una o più componenti fisiche necessarie per un progetto hardware.	PROGETTO_HARDWARE, COMPONENTE	
<b>PROFILO_PROGETTO</b>	La lista di uno o più profili necessari per lo sviluppo di un progetto software.	PROGETTO_SOFTWARE, PROFILO	
<b>SKILL_PROFILo</b>	La lista di una o più skill comprese all'interno di un profilo di sviluppo, e il livello richiesto per ciascuna di esse.	SKILL, PROFILO	livello_richies

N.B. **PARTECIPANTE**.stato → enum: {"accettato", "rifiutato", "potenziale"}

## 2.4. BUSINESS RULES

REGOLE DI VINCOLO	
1.	Per le <b>skill di curriculum</b> , il <b>livello è un numero tra 0 e 5</b>
2.	Per le <b>skill di profilo</b> , il <b>livello è un numero tra 0 e 5</b>
3.	<b>Ogni componente</b> in un progetto <b>hardware</b> dispone di una <b>quantità maggiore di zero</b>
4.	<b>Solo</b> gli utenti <b>amministratori</b> possono <b>popolare la lista delle competenze</b>
5.	Se la <b>somma totale degli importi dei finanziamenti supera il budget</b> del progetto, lo stato di tale progetto <b>diventa pari a chiuso</b>
6.	Se il <b>progetto resta in stato aperto oltre la data limite</b> , lo stato di tale progetto <b>diventa pari a chiuso</b>
7.	Un <b>progetto chiuso</b> non accetta <b>ulteriori finanziamenti</b>
8.	Ogni commento ha al <b>massimo 1 risposta</b> scritta dall' <b>utente creatore del progetto</b>
9.	La piattaforma consente ad un utente di inserire una candidatura su un profilo <b>SOLO se, per ogni skill richiesta da un profilo, l'utente dispone di un livello superiore o uguale</b> al valore richiesto
10.	La <b>reward</b> ottenuta da un <b>finanziamento</b> dipende dal suo <b>importo</b>
11.	Il <b>budget di un progetto hardware</b> non può essere <b>inferiore alla somma del costo delle sue componenti</b>

## 3. PROGETTAZIONE LOGICA

---

### 3.1. ANALISI DELLE RIDONDANZE

Si vuole valutare se la seguente ridondanza: campo `nr_progetti` relativo ad un utente creatore debba essere **tenuta o eliminata**, sulla base delle seguenti operazioni:

#### Operazioni

- $Op_1$ : **Aggiungere (write)** un nuovo progetto ad un utente creatore esistente (**1 volta/mese, interattiva**)
- $Op_2$ : **Visualizzare (read)** tutti i progetti e tutti i finanziamenti (**1 volta/mese, batch**)
- $Op_3$ : **Contare (read)** il numero di progetti associati ad uno specifico utente (**3 volte/mese, batch**)

#### Contesto

Si deriva il **costo**  $c$  di un'operazione  $O_T$ , dunque  $c(O_T)$ , utilizzando la seguente formula:

$$c(O_T) = f(O_T) \cdot w_T \cdot (\alpha \cdot NC_{\text{write}} + NC_{\text{read}})$$

Dove:

- $f(O_T)$  = **Frequenza** dell'operazione
- $NC_{\text{write}}$  = Numero di **accessi in scrittura** a componenti (entità/relazioni) dello schema
- $NC_{\text{read}}$  = Numero di **accessi in lettura** a componenti (entità/relazioni) dello schema
- $w_T$  = **Peso** dell'operazione (interattiva/batch)
- $\alpha$  = **Coefficiente moltiplicativo** delle operazioni in **scrittura**

## Coefficienti per l'Analisi

- $\alpha = 2$
- $w_I = 1$
- $w_B = 0.5$

## Tabella dei Volumi

- 10 progetti
- 3 finanziamenti per progetto
- 5 utenti
- 2 progetti per utente

## Operazione 1

---

$Op_1$ : **Aggiungere (write)** un nuovo progetto ad un utente creatore esistente (**1 volte/mese, interattiva**).

Includendo `nr_progetti`

### Logica

1. Incrementa di uno `nr_progetti` dell'utente creatore

2. Crea un nuovo progetto
  - Una entry per il progetto
  - Una o più entry per le foto del progetto
  - Una o più entry per le reward del progetto
3. Inserisci una entry nella tabella relativa al tipo di progetto (software/hardware)
  - IF software → Una o più entry per i profili richiesti
  - IF hardware → Una o più entry per le componenti richieste

**Procedura** (assumendo una foto, una reward, e un profilo/componente)

- 1 UPDATE in CREATORE.nr\_progetti
- 1 INSERT in PROGETTO
- 1 INSERT in FOTO
- 1 INSERT in REWARD
- 1 INSERT in PROGETTO\_SOFTWARE / PROGETTO\_HARDWARE
- 1 INSERT in PROFILO / COMPONENTE

### Variabili

- $f(Op_1) = 1$
- $w_I = 1$
- $\alpha = 2$
- $NC_{\text{write}} = 6$
- $NC_{\text{read}} = 0$

### Costo

$$c_1(Op_1) = 1 \cdot 1 \cdot (2 \cdot 6 + 0) = 12$$

Escludendo nr\_progetti

### Logica

- Identica al caso di sopra eccetto per il primo passo

### Procedura

- 1 INSERT in PROGETTO
- 1 INSERT in FOTO

- 1 INSERT in REWARD
- 1 INSERT in PROGETTO\_SOFTWARE / PROGETTO\_HARDWARE
- 1 INSERT in PROFILO / COMPONENTE

## Variabili

- Identiche al caso di sopra eccetto:
- $NC_{\text{write}} = 5$

## Costo

$$c_2(Op_1) = 1 \cdot 1 \cdot (2 \cdot 5 + 0) = 10$$

## Operazione 2

---

$Op_2$ : **Visualizzare (read)** tutti i progetti e tutti i finanziamenti (**1 volta/mese, batch**).

Includendo `nr_progetti`

## Logica

1. Leggi l'intera tabella PROGETTO
2. Leggi l'intera tabella FINANZIAMENTO

## Procedura

- 10 SELECT in PROGETTO
- 30 SELECT in FINANZIAMENTO

## Variabili

- $f(Op_1) = 1$
- $w_B = 0.5$
- $\alpha = 2$
- $NC_{\text{write}} = 0$
- $NC_{\text{read}} = 40$

## Costo

$$c_1(Op_2) = 1 \cdot 0.5 \cdot (2 \cdot 0 + 40) = 20$$

Escludendo `nr_progetti`

I calcoli rimangono invariati a quelli fatti di sopra.

Costo

$$c_2(Op_2) = 1 \cdot 0.5 \cdot (2 \cdot 0 + 40) = 20$$

## Operazione 3

---

$Op_3$ : **Contare (read)** il numero di progetti associati ad uno specifico utente (**3 volte/mese, batch**).

Includendo `nr_progetti`

Logica

1. Leggi l'attributo `nr_progetti` dell'utente.

**Procedura** (assumendo che per "associati" si intenda progetti creati da un utente)

- 1 `SELECT` in `CREATORE`

Variabili

- $f(Op_1) = 3$
- $w_B = 0.5$
- $\alpha = 2$
- $NC_{\text{write}} = 0$
- $NC_{\text{read}} = 1$

Costo

$$c_1(Op_3) = 3 \cdot 0.5 \cdot (2 \cdot 0 + 1) = 1.5$$

Escludendo `nr_progetti`

## Logica (assumendo l'assenza di un indice efficiente)

1. Leggi l'intera tabella PROGETTO
2. Filtra laddove l'email del creatore del progetto corrente non corrisponde all'email del creatore per cui si sta facendo la query

## Procedura

- 2 SELECT in PROGETTO , avendo scandito **10 entry**

## Variabili

- $f(Op_1) = 3$
- $w_B = 0.5$
- $\alpha = 2$
- $NC_{\text{write}} = 0$
- $NC_{\text{read}} = 10$

## Costo

$$c_2(Op_3) = 3 \cdot 0.5 \cdot (2 \cdot 0 + 10) = 15$$

## Conclusione

---

Includendo **nr\_progetti**

$$\sum_{T=1}^3 c_1(Op_T) = 12 + 20 + 1.5 = 33.5$$

Escludendo **nr\_progetti**

$$\sum_{T=1}^3 c_2(Op_T) = 10 + 20 + 15 = 45$$

## Speedup

$$\frac{45}{33.5} = 1.34$$

## Analisi

	Includendo <code>nr_progetti</code>	Escludendo <code>nr_progetti</code>
$Op_1$	12	10
$Op_2$	20	20
$Op_3$	1.5	15
<b>Totale</b>	33.5	45

Osservando i costi di entrambi scenari, risulta che **includere `nr_progetti` sia l'approccio corretto**. Il costo per le prime due operazione è praticamente identico. Il guadagno principale nel mantenere `nr_progetti` deriva dalla differenza di costo per la terza ed ultima operazione,  $Op_3$ , che richiede di contare il numero di progetti associati ad un utente creatore nella piattaforma. Lo **speedup** infatti è **pari ad 1.34**, indicando un **guadagno di efficienza circa del 34% includendo la ridondanza** sulla base delle operazioni elencate.

**N.B.** Anche se ci fosse per l'ultima operazione un indice efficiente che tenga traccia dell'associazione tra progetti e i creatori di essi, allora l'assenza della ridondanza comporterebbe un costo di 3 (basta porre  $NC_{read} = 2$  invece che 10), che è comunque **2x più costoso rispetto all'utilizzo della ridondanza** (costo di 1.5). Tenendo questa considerazione a mente, man mano che **la piattaforma cresce e più utenti creatori creano e gestiscono più progetti**, la differenza (di 2x) diventa sempre più notevole, e pertanto **converrà comunque mantenere la ridondanza**.

**Includendo `nr_progetti`**, l'operazione è immediata, dovendo semplicemente leggere l'attributo `CREATORE.nr_progetti` dell'utente. Infatti, per  $Op_3$ , il costo è 10 volte minore tenendo conto di `nr_progetti`, ed il costo in memoria associato è trascurabile (tenendo conto la tabella di volumi fornita nella traccia), che presuppone in media 2 progetti per utente, e 5 utenti sulla piattaforma. Volendo ottimizzare ulteriormente il costo in memoria di `nr_progetti`, lo si può rappresentare in MySQL come un `TINYINT UNSIGNED`, che ha un costo di 1 byte e può rappresentare valori compresi in  $[0, 255]$  (valori negativi non hanno senso in questo contesto). In pratica però non ha senso un'ottimizzazione del genere, dunque lo rappresento come un `INT UNSIGNED` con un costo di 4 byte, ma un range di valori in  $[0, 2^{32} - 1]$ .

**Escludendo `nr_progetti`**, l'operazione, a differenza dello scenario di sopra, non ha accesso a `CREATORE.nr_progetti`, e deve pertanto scandire ogni progetto nella tabella `PROGETTO`, effettivamente leggendo ogni entry (10 in tutto), e verificando su ciascuna se l'email del creatore combacia con quella inserita nella query.

## 3.2. LISTA DELLE TABELLE

**UTENTE**(email, password, nickname, nome, cognome, anno\_nascita, luogo\_nascita)

**ADMIN**(email\_utente, codice\_sicurezza)

**CREATORE**(email\_utente, nr\_progetti, affidabilità)

**PROGETTO**(nome, email\_creatore, descrizione, budget, stato, data\_inserimento, data\_limite)

**FOTO**(id, nome\_progetto, foto)

**REWARD**(codice, nome\_progetto, descrizione, foto, min\_importo)

**COMMENTO**(id, email\_utente, nome\_progetto, data, testo, risposta)

**PROGETTO\_SOFTWARE**(nome\_progetto)

**PROGETTO\_HARDWARE**(nome\_progetto)

**COMPONENTE**(nome\_componente, nome\_progetto, descrizione, quantità, prezzo)

**PROFILO**(nome\_profilo, nome\_progetto)

**SKILL**(competenza)

**FINANZIAMENTO**(data, email\_utente, nome\_progetto, codice\_reward, importo)

**SKILL\_CURRICULUM**(email\_utente, competenza, livello\_effettivo)

**SKILL\_PROFILo**(nome\_profilo, competenza, nome\_progetto, livello\_richiesto)

**PARTECIPANTE**(email\_utente, nome\_progetto, nome\_profilo, stato)

## 3.3. LISTA DEI VINCOLI INTER-RELAZIONALI

**ADMIN.email\_utente** → **UTENTE.email**

**CREATORE.email\_utente** → **UTENTE.email**

**PROGETTO**.email\_creatore → **UTENTE**.email  
**COMMENTO**.email\_utente → **UTENTE**.email  
**FINANZIAMENTO**.email\_utente → **UTENTE**.email  
**PARTECIPANTE**.email\_utente → **UTENTE**.email  
**SKILL\_CURRICULUM**.email\_utente → **UTENTE**.email

**FOTO**.nome\_progetto → **PROGETTO**.nome  
**REWARD**.nome\_progetto → **PROGETTO**.nome  
**COMMENTO**.nome\_progetto → **PROGETTO**.nome  
**PROGETTO\_SOFTWARE**.nome\_progetto → **PROGETTO**.nome  
**PROGETTO\_HARDWARE**.nome\_progetto → **PROGETTO**.nome  
**COMPONENTE**.nome\_progetto → **PROGETTO**.nome  
**FINANZIAMENTO**.nome\_progetto → **PROGETTO**.nome  
**PARTECIPANTE**.nome\_progetto → **PROGETTO**.nome  
**PROFILO**.nome\_progetto → **PROGETTO**.nome

**FINANZIAMENTO**.codice\_reward → **REWARD**.codice

**SKILL\_PROFILE**.nome\_profilo → **PROFILO**.nome\_profilo  
**PARTECIPANTE**.nome\_profilo → **PROFILO**.nome\_profilo

**SKILL\_CURRICULUM**.competenza → **SKILL**.competenza  
**SKILL\_PROFILE**.competenza → **SKILL**.competenza

## 4. NORMALIZZAZIONE

---

### 4.1. ANALISI

In questa sezione viene analizzato lo schema logico prodotto sulla base della terza, e FNBC forma normale.

#### 3FN

Solo se per ogni dipendenza funzionale X→Y:

- X è una superchiave dello schema  
**Oppure**
- Y appartiene ad una chiave candidata dello schema

# FNBC

Solo se per ogni dipendenza funzionale  $X \rightarrow Y$ :

- $X$  è una superchiave dello schema

Di seguito viene dimostrato che **ogni tabella proposta di sopra è in Forma Normale Boyce & Codd.**

## UTENTE

- $R(\text{email}, \text{password}, \text{nickname}, \text{nome}, \text{cognome}, \text{anno_nascita}, \text{luogo_nascita})$
- $F = \{\text{email} \rightarrow \text{OGNI ATTRIBUTO}\}$
- 3FN:  / FNBC:

## ADMIN

- $R(\text{email_utente}, \text{codice_sicurezza})$
- $F = \{\text{email_utente} \rightarrow \text{codice_sicurezza}\}$
- 3FN:  / FNBC:

## CREATORE

- $R(\text{email_utente}, \text{nr_progetti}, \text{affidabilità})$
- $F = \{\text{email_utente} \rightarrow \text{nr_progetti}, \text{affidabilità}\}$
- 3FN:  / FNBC:

## PROGETTO

- $R(\text{nome}, \text{email_creatore}, \text{descrizione}, \text{budget}, \text{stato}, \text{data_inserimento}, \text{data_limite})$
- $F = \{\text{nome} \rightarrow \text{OGNI ATTRIBUTO}\}$
- 3FN:  / FNBC:

## FOTO

- $R(\text{id}, \text{nome_progetto}, \text{foto})$
- $F = \{\text{id}, \text{nome_progetto} \rightarrow \text{foto}\}$
- 3FN:  / FNBC:

## REWARD

- **R**(codice, nome\_progetto, descrizione, foto, min\_importo)
- **F** = {codice, nome\_progetto → OGNI ATTRIBUTO}
- **3FN:** / **FNBC:**

## COMMENTO

- **R**(id, email\_utente, nome\_progetto, data, testo, risposta)
- **F** = {id → OGNI ATTRIBUTO}
- **3FN:** / **FNBC:**

## PROGETTO\_SOFTWARE

- **R**(nome\_progetto)
- **F** = DF Banale
- **3FN:** / **FNBC:**

## PROGETTO\_HARDWARE

- **R**(nome\_progetto)
- **F** = DF Banale
- **3FN:** / **FNBC:**

## COMPONENTE

- **R**(nome\_componente, nome\_progetto, descrizione, quantità, prezzo)
- **F** = {nome\_componente, nome\_progetto → OGNI ATTRIBUTO}
- **3FN:** / **FNBC:**

## PROFILO

- **R**(nome\_profilo, nome\_progetto)
- **F** = DF Banale
- **3FN:** / **FNBC:**

## SKILL

- **R**(competenza)

- **F** = DF Banale
- 3FN: / FNBC:

## FINANZIAMENTO

- **R**(data, email\_utente, nome\_progetto, codice\_reward, importo)
- **F** = {data, email\_utente, nome\_progetto → OGNI ATTRIBUTO}
- 3FN: / FNBC:

## SKILL\_CURRICULUM

- **R**(email\_utente, competenza, livello\_effettivo)
- **F** = {email\_utente, competenza → livello\_effettivo}
- 3FN: / FNBC:

## SKILL\_PROFIL0

- **R**(nome\_profilo, competenza, nome\_progetto, livello Richiesto)
- **F** = {nome\_profilo, competenza, nome\_progetto → livello Richiesto}
- 3FN: / FNBC:

## PARTECIPANTE

- **R**(email\_utente, nome\_progetto, nome\_profilo stato)
- **F** = {email\_utente, nome\_progetto, nome\_profilo → stato}
- 3FN: / FNBC:

# 5. FUNZIONALITÀ

---

## PREREQUISITI

### Requisiti Software

- **Docker e Docker Compose**: Per la containerizzazione e l'orchestrazione
- **MySQL 8.0+**: Database per i dati principali dell'applicazione
- **MongoDB**: Database secondario per la funzionalità di logging

- **PHP 8.2** con estensioni:
  - `pdo_mysql` : Per la connettività MySQL
  - `mongodb` : Per la connettività MongoDB
- **Apache Web Server**: Per servire l'applicazione PHP

## Requisiti di Sistema

- Porta `8080` disponibile per l'accesso web
- Porte `3307` e `27017` disponibili per l'accesso ai database

## 5.1. BACKEND (MySQL)

### INIZIALIZZAZIONE

Il file di inizializzazione del database, `01-bostarter_init.sql`, si suddivide nelle seguenti parti principali:

#### TABELLE

- Tutte le tabelle usate dal database.

#### STORED PROCEDURES (HELPER)

- Tutte le stored procedure di tipo secondario/helper utilizzate da altre stored procedure (primarie/main) per effettuare controlli di sicurezza, o a livello di applicazione invocate mediante la funzione mia `sp_invoke(...)`.

#### STORED PROCEDURES (MAIN)

- Tutte le stored procedures di tipo primario/main, fungendo come interfaccia principale fra l'applicazione (sempre mediante `sp_invoke(...)`) e il database per quasi ogni operazioni disponibile sulla piattaforma.

#### VISTE

- Le tre viste richieste dal progetto.

#### TRIGGERS

- Tutti i trigger richiesti dal progetto.

## EVENTI

- Il solo evento richiesto dal progetto.

Le tabelle, attraverso vincoli inter-relazionali e check definiti a livello di attributo, fungono già come un buon punto di partenza in termini di sicurezza e consistenza dei dati. Ho optato, però, di implementare un ulteriore livello di sicurezza centrale e robusto, all'interno delle stored procedures definite nel file.

Come menzionato di sopra, ho suddiviso le stored procedures in due categorie principali: Main e helper. Le stored procedure "main" performano operazioni richieste dalla piattaforma (es. aggiunta di una skill globale), e si appoggiano a stored procedures di tipo "helper" per verificare che ogni controllo di sicurezza richiesto per l'operazione sia garantito (es. l'utente che aggiunge una skill globale deve essere admin).

## POPOLAMENTO

Nel file `02-bostarter_demo.sql` vengono fatte chiamate delle stored procedures definite nel file di sopra per inserire i dati fintizi nella piattaforma. Ovviamente per il suo corretto funzionamento vengono fatte solo chiamate valide, e non vengono utilizzate qui stored procedures che rimuovono/aggiornano dati (ha più senso fare una dimostrazione di esso in sede d'esame).

Inoltre, viene automaticamente invocato lo script `config/seed_data.php` che popola i rimanenti dati necessari per la demo della piattaforma, che non potevano essere popolati dal file sql.

## 5.2. FRONTEND (PHP)

### STRUTTURA GENERALE

```
bostarter/
└── actions/
    ├── candidatura_insert.php
    ├── candidatura_update.php
    ├── commento_delete.php
    ├── commento_insert.php
    ├── commento_risposta_delete.php
    ├── commento_risposta_insert.php
    └── componente_delete.php
```

```
|   └── componente_insert.php  
|   └── componente_update.php  
|   └── finanziamento_insert.php  
|   └── foto_delete.php  
|   └── foto_insert.php  
|   └── login_handler.php  
|   └── logout.php  
|   └── profilo_delete.php  
|   └── profilo_insert.php  
|   └── profilo_nome_update.php  
|   └── progetto_budget_update.php  
|   └── progetto_descrizione_update.php  
|   └── progetto_insert.php  
|   └── register_handler.php  
|   └── reward_insert.php  
|   └── skill_curriculum_delete.php  
|   └── skill_curriculum_insert.php  
|   └── skill_curriculum_update.php  
|   └── skill_insert.php  
|   └── skill_profilo_delete.php  
|   └── skill_profilo_insert.php  
|   └── skill_profilo_update.php  
|   └── skill_update.php  
|   └── utente_convert_creatore.php  
└── components/  
    ├── componente_modifica.php  
    ├── componente_nuovo.php  
    ├── componenti_esistenti.php  
    ├── error_alert.php  
    ├── footer.php  
    ├── header.php  
    ├── profili_esistenti.php  
    ├── profilo_modifica.php  
    ├── profilo_nuovo.php  
    ├── profilo_skill_modifica.php  
    ├── progetto_aggiorna_budget.php  
    ├── progetto_aggiorna_componenti.php  
    ├── progetto_aggiorna_descrizione.php  
    ├── progetto_aggiorna_profilo.php  
    ├── progetto_aggiorna_reward.php  
    └── success_alert.php  
└── config/  
    └── config.php
```

```
|   └── seed_data.php
|   └── functions/
|       ├── checks.php
|       ├── EventPipeline.php
|       ├── helpers.php
|       ├── log.php
|       ├── redirect.php
|       ├── sp_invoke.php
|       └── url.php
└── public/
    ├── libs/
    ├── candidature.php
    ├── componente_conferma_insert.php
    ├── componente_conferma_update.php
    ├── curriculum.php
    ├── curriculum_skill_global_update.php
    ├── curriculum_skill_update.php
    ├── finanziamenti.php
    ├── finanziamento_conferma.php
    ├── home.php
    ├── index.php
    ├── login.php
    ├── logs.php
    ├── progetti.php
    ├── progetto_aggiorna.php
    ├── progetto_crea.php
    ├── progetto_dettagli.php
    ├── register.php
    └── statistiche.php
```

## /actions

- Le operazioni più complesse (generalmente associate ad una o più "main" stored procedures) che permettono il funzionamento della piattaforma. Vengono invocate all'interno delle pagine mediante i form.

## /components

- Componenti grafiche individuali, utilizzate in pagine /public per delineare più chiaramente la struttura delle pagine

## /config

- Contiene le configurazioni necessarie per la piattaforma, come connessione al database, ed alcuni dati pre-caricati per la demo.

## /functions

- Funzioni primitive / semi-primitive che eseguono controlli od operazioni semplici e comuni, usate per il funzionamento delle pagine.

## /public

- Le pagine php visibili al client, fungono come interfaccia principale per gli utenti nell'interazione con la piattaforma e database.

# STRUTTURA FILE

## Directory /public

I file nella directory `/public` seguono un'architettura modulare con sezioni chiaramente delimitate:

- `==> SETUP ==>` : Inizializzazione della sessione, caricamento delle dipendenze e verifica dell'autenticazione
- `==> VALIDATION ==>` : Controlli di sicurezza specifici per la pagina corrente
- `==> DATA ==>` : Recupero dei dati dal database tramite stored procedures
- `==> RENDERING ==>` : Funzioni per la generazione di componenti UI riutilizzabili

Questa struttura favorisce la separazione tra logica applicativa e presentazione, con inclusione di componenti HTML e minimo codice PHP inline per il rendering condizionale. La pagina utilizza i dati recuperati per costruire l'interfaccia utente, delegando la manipolazione dei dati alle stored procedures. In maniera analoga anche i file in `/components` seguono la stessa struttura, essendo componenti grafiche individuali che vengono importate nelle pagine di `/public`.

## Directory /actions

I file nella directory `/actions` implementano un pattern pipeline uniforme:

- **==> SETUP** : Inizializzazione della sessione, caricamento delle dipendenze e verifica dell'autenticazione
- **==> VARIABLES** : Estrazione e validazione dei parametri di input necessari per l'operazione
- **==> CONTEXT** : Creazione di un contesto operativo (collezione, azione, redirect, procedura, ecc.)
- **==> VALIDATION** : Controlli di validazione con interruzione al primo errore
- **==> ACTION** : Invocazione della stored procedure per l'operazione sul database
- **==> SUCCESS** : Gestione del successo con logging e redirect appropriato (definiti nella sezione di contesto sopra)

Questa architettura centralizza la gestione degli errori e il logging nella classe `EventPipeline`, eliminando la necessità di recupero dati o rendering HTML poiché questi file eseguono operazioni atomiche sul database con redirect automatico. Il pattern fornisce un approccio consistente alla validazione dei dati, gestione delle transazioni e feedback all'utente.

## Directory `/functions`

I file nella directory `/functions` implementano funzionalità di supporto e utility condivise nell'intero sistema. Questa directory contiene moduli specializzati come `sp_invoke.php`, che fornisce un'interfaccia semplificata per l'invocazione delle stored procedure MySQL, e `EventPipeline.php`, che implementa il pattern pipeline per gestire in modo consistente validazioni, operazioni sul database e logging. Altri componenti chiave includono `helpers.php` con funzioni di utilità generale, `checks.php` per i controlli di sicurezza, `redirect.php` per la gestione dei reindirizzamenti, e `url.php` per la generazione centralizzata degli URL.

Questa organizzazione favorisce la riusabilità del codice e la separazione delle responsabilità, permettendo ai file delle directory `/public` e `/actions` di delegare le operazioni comuni a funzioni specializzate. L'approccio riduce la duplicazione del codice e aumenta la coerenza dell'implementazione, facilitando la manutenzione e l'estensione della piattaforma.

## AUTENTICAZIONE UTENTE

- `login.php`
- `register.php`

Il landing page della piattaforma, `index.php`, verifica se l'utente si è già autenticato controllando la variabile di sessione `$_SESSION['user_email']` e se sì allora viene

reindirizzato alla homepage, `home.php`, altrimenti viene reindirizzato alla pagina di login, `login.php` per autenticarsi.

Se dispone di un account esistente sulla piattaforma (email e password) allora può autenticarsi, altrimenti clicca su Registra e continua con la procedura per creare il proprio account. In fase di login l'utente può anche autenticarsi come amministratore se dispone del codice di sicurezza proprio.

The screenshot shows the BOSTARTER login interface. At the top left is the BOSTARTER logo, and at the top right are 'Login' and 'Register' buttons. The main area is titled 'Login' and contains three input fields: 'Email' (placeholder: 'Inserisci il tuo indirizzo email'), 'Password' (placeholder: 'Inserisci la tua password'), and 'Codice di Sicurezza (ADMIN ONLY)' (placeholder: 'Inserisci il codice di sicurezza'). Below these is a blue 'Login' button. At the bottom of the form is a link 'Non hai un account? [Registrati](#)'.

Verrà resa disponibile in sede d'esame una sezione di autologin per poter passare rapidamente fra utenti e testare diverse funzionalità della piattaforma.



La struttura della pagina di registrazione è molto simile a quella di login per mantenere un look consistente e prevedibile per l'utente, con la sola differenza che contiene alcuni campi aggiuntivi per i propri dati personali.

### Registrazione

Email

Nickname

Nome

Cognome

Anno di Nascita

Luogo di Nascita

Password

Conferma Password

Sei un creatore di progetti?

Sei un amministratore?

Codice di Sicurezza (ADMIN ONLY)

Registrati

Hai già un account? [Accedi](#)

© 2025 BOSTARTER. All rights reserved.

In fase di registrazione, l'utente ha la possibilità di segnarsi come un creatore e/o admin della piattaforma, cliccando sulle checkbox sopra al submit button della registrazione. In tal caso, verrà inserito nel DB anche come CREATORE / ADMIN .

## HOME

- [home .php](#)

La pagina di home funge come luogo centrale della piattaforma, dove l'utente ha modo di visualizzare alcune informazioni del proprio account, e di recarsi in altre sezioni specifiche del sito. Il metodo principale di navigazione fra sezioni principali della piattaforma è mediante la navbar, ed all'interno di ciascuna sezione si hanno bottoni specifici che reindirizzano in sottopagine del sito.

### Il tuo profilo

Dati personali		Creatore
Nome: Bob	Cognome: Bianchi	
Email: bob@example.com	Nickname: bobby85	
Anno di nascita: 1985	Luogo di nascita: Roma	

1 Competenze	0 Candidature	2 Finanziamenti
JavaScript	Nessuna candidatura effettuata	Total: 1,600.00 €
1 Progetti	2 Partecipanti	100.00% Affidabilità

### I tuoi progetti

ProgettoAlpha	Budget: 10,000.00€	APERTO
---------------	--------------------	--------

### Finanziamenti recenti

ProgettoBeta	Reward: RWD3_Beta	1,600.00€
		2025-03-22

[Visualizza tutti](#)

© 2025 BOSTARTER. All rights reserved.

## STATISTICHE

- statistiche.php

La pagina delle statistiche è relativamente semplice, chiamando i dati presenti nelle 3 viste definite nel database, e rendendole visibili nella forma di tabelle.

### Statistiche

Top 3 Creatori (Affidabilità)	
Classifica dei top 3 utenti creatori, in base al loro valore di affidabilità.	
1. bobby85	
2. diana12	
3. gracieee	

Top 3 Progetti (Completamento)	
Classifica dei top 3 progetti aperti più vicini al completamento del budget.	
1. ProgettoBeta	5550.00 / 12850.00€ 43.19%
2. ProgettoAlpha	3884.00 / 10000.00€ 38.84%
3. ProgettoDelta	3087.00 / 8000.00€ 38.59%

Top 3 Utenti (Finanziamenti)	
Classifica dei top 3 utenti, in base al totale dei finanziamenti erogati.	
1. oscar	
2. edo_unibo	
3. frankenstein	

© 2025 BOSTARTER. All rights reserved.

Non sono previste alcune operazioni da nessun utente su questa pagina; i dati prodotti dalle viste vengono aggiornati in tempo reale, ricaricando la pagina.

# CURRICULUM

- [curriculum.php](#)

La pagina curriculum consente agli utenti di gestire le proprie competenze professionali. Ogni utente può aggiungere skill selezionandole dalla lista globale e specificando il proprio livello di competenza (da 0 a 5). L'utente può anche modificare (il livello) o rimuovere proprie skill esistenti.

The screenshot shows the 'Curriculum' section of the BOSTARTER platform. At the top, there's a navigation bar with links for 'BOSTARTER', 'Progetti', 'Finanziamenti', 'Statistiche', 'Curriculum', 'Candidature', and 'Logout (bob@example.com)'. Below the navigation is a blue header 'Le mie Skill' containing a message: 'Queste sono le skill attualmente associate al tuo profilo.' A table lists three skills: Java (level 5/5), JavaScript (level 3/5), and MySQL (level 4/5), each with 'Modifica' and 'Rimuovi' buttons. Below this is a green header 'Aggiungi Skill' with a message: 'Seleziona una skill disponibile e inserisci il livello che possiedi.' It features a dropdown for 'Skill Disponibile' and a slider for 'Livello (0-5)'. A red footer bar at the bottom contains the text '© 2025 BOSTARTER. All rights reserved.'

Per gli amministratori, è disponibile una sezione aggiuntiva per gestire la lista globale delle competenze disponibili sulla piattaforma. Questa funzionalità permette di aggiungere nuove competenze o modificare (il nome) quelle esistenti.

The screenshot shows the 'Gestisci Skill (Admin)' section. It has a red header 'Aggiungi o modifica le skill disponibili globalmente per tutti gli utenti.' A 'Nuova Skill Globale' form includes a text input 'Inserisci la nuova skill' and a red 'Aggiungi Skill Globale' button. Below is a table titled 'Skill Globali' listing five skills: Angular, C++, CSS, ERD, and Figma, each with a 'Modifica' button. A red footer bar at the bottom contains the text '© 2025 BOSTARTER. All rights reserved.'

# FINANZIAMENTI

- `finanziamenti.php`
- `finanziamento_conferma.php`

La pagina dei finanziamenti mostra la cronologia di tutti i finanziamenti effettuati dall'utente. Per gli utenti creatori, è presente anche una sezione che visualizza i finanziamenti ricevuti dai propri progetti.

BOSTARTER    [Progetti](#) [Finanziamenti](#) [Statistiche](#) [Curriculum](#) [Candidature](#)    [Logout \(bob@example.com\)](#)

## Finanziamenti

Finanziamenti Ricevuti					Total: 3,884.00€
Data	Progetto	Finanziatore	Importo	Reward	
2025-03-22	<a href="#">ProgettoAlpha</a>	charlie_chaplin charlie@example.com	1,150.00€	 <b>RWD1_Alpha</b> Do deserunt ullamco aliquip ad consequat Lorem minim irure.	
2025-03-22	<a href="#">ProgettoAlpha</a>	diana12 diana@example.com	1,211.00€	 <b>RWD2_Alpha</b> Occaecat do elit consequat esse voluptate cillum fugiat.	
2025-03-22	<a href="#">ProgettoAlpha</a>	oscar oscar@example.com	1,523.00€	 <b>RWD3_Alpha</b> Aliquip ea duis excepteur dolor elit pridient ipsum qui.	

Finanziamenti Effettuati					Total: 1,600.00€
Data	Progetto	Creatore	Importo	Reward	
2025-03-22	<a href="#">ProgettoBeta</a>	diana@example.com	1,600.00€	 <b>RWD3_Beta</b> Magna tempor in amet sit incididunt magna elit nostrud laborum in.	

© 2025 BOSTARTER. All rights reserved.

Quando un utente decide di finanziare un progetto dalla pagina di dettaglio, viene reindirizzato a `finanziamento_conferma.php` dove può selezionare una reward tra quelle disponibili in base all'importo che intende donare.

# PROGETTI

- `progetti.php`
- `progetto_crea.php`

La pagina progetti visualizza tutti i progetti disponibili sulla piattaforma, con informazioni riassuntive su ciascuno: nome, tipo (software/hardware), stato (aperto/chiuso), percentuale di completamento, e giorni rimanenti alla chiusura.

## Progetti

[Crea Progetto](#)
**ProgettoAlpha**  
SOFTWARE
 

APERTO

Creatore: bob@example.com

Budget: 10,000.00€

Molit eiusmod deserunt amet do eu anim ipsum sit. Fugiat consectetur aute duis eiusmod adipisicing quis laboris in. Consectetur voluptate cupidatat ipsum id elit.

38.84%

3,884.00€ / 10,000.00€

Durata: 22/03/2025 - 31/12/2025

283 GIORNI RIMASTI

**ProgettoBeta**  
HARDWARE
 

APERTO

Creatore: diana@example.com

Budget: 12,850.00€

Sint consequat officia anim eu voluptate Lorem. velit sint labore excepteur laboris magna do esse. Irure ut aliquip sit aliquip esse qui proident culpa fugiat est elit veniam deserunt.

43.19%

5,550.00€ / 12,850.00€

Durata: 22/03/2025 - 30/11/2025

252 GIORNI RIMASTI

**ProgettoDelta**  
HARDWARE
 

APERTO

Creatore: ivan@example.com

Budget: 8,000.00€

Incididunt reprehenderit velit irure Lorem do ipsum tempor reprehenderit magna ut dolore sint est incididunt cùis. Sit officia irure cillum do. Tempor minim minim ut et molit et eu adipisicing non.

38.59%

3,017.00€ / 8,000.00€

Durata: 22/03/2025 - 12/06/2025

81 GIORNI RIMASTI

**ProgettoEpsilon**  
SOFTWARE
 

CHIUSO

Creatore: karen@example.com

Budget: 12,000.00€

Et velit aliqua ipsum nisi. Qui sunt consectetur eiusmod consequat tempor id aute proident velit velit. Magna cillum et qui molit sint aliquip aute anim elit. Consectetur veniam in do amet occaecat nostrud aliquip.

100%

12,000.00€ / 12,000.00€

Durata: 22/03/2025 - 31/05/2025

TERMINATO

**ProgettoGamma**  
SOFTWARE
 

APERTO

Creatore: grace@example.com

Budget: 15,000.00€

Sunt voluptate aliqua laboris voluptate adipisicing voluptate molit do ut aute magna ad. Deserunt nostrud cupidatat ullamco in reprehenderit ex ut in.

21.33%

3,200.00€ / 15,000.00€

Durata: 22/03/2025 - 16/01/2026

299 GIORNI RIMASTI

**ProgettoKappa**  
HARDWARE
 

APERTO

Creatore: ivan@example.com

Budget: 7,500.00€

Magna laborum mollit incididunt officia non elit molit minim. Cupiditat adipisicing esse sit occaecat. Culpa mollit non aliquip reprehenderit aute eiusmod aliqua nisi aute pariatur mollit.

16.16%

1,212.00€ / 7,500.00€

Durata: 22/03/2025 - 31/07/2025

130 GIORNI RIMASTI

© 2025 BOSTARTER. All rights reserved.

Gli utenti creatori visualizzano anche un bottone "Crea Progetto" che consente loro di creare un nuovo progetto, mentre gli utenti regolari vedono un bottone "Diventa Creatore" che permette loro di acquisire il ruolo di creatore sulla piattaforma.

La pagina `progetto_crea.php` guida il creatore attraverso il processo di inserimento dei dati del progetto: nome, descrizione, budget, data limite e tipo (software/hardware). Il creatore ha modo di inserire ulteriori informazioni riguardanti il progetto nell'apposita pagina di dettaglio (vedi sezione successiva).

## Crea Nuovo Progetto

**Informazioni Progetto**

## Nome Progetto

Il nome del progetto deve essere unico.

## Descrizione

Descrivi il tuo progetto in dettaglio.

## Budget (€)

L'importo totale necessario per il progetto.

## Data Limite

La data limite per il progetto. Deve essere futura ad oggi.

## Tipo di Progetto

 Software
 
 Hardware
 

Seleziona il tipo di progetto che stai creando.

[Crea Progetto](#)

© 2025 BOSTARTER. All rights reserved.

# DETTAGLI PROGETTO

- `progetto_dettagli.php`

La pagina di dettaglio progetto è il centro operativo dove convergono la maggior parte delle funzionalità della piattaforma. Visualizza tutte le informazioni relative al progetto selezionato: descrizione, foto, budget, somma finanziamenti ricevuti, reward disponibili, componenti o profili richiesti in base al tipo, ed in fondo i commenti del progetto.

BOSTARTER    [Progetti](#) [Finanziamenti](#) [Statistiche](#) [Curriculum](#) [Candidature](#)    [Logout \(bob@example.com\)](#)

**ProgettoAlpha**  
SOFTWARE

**APERTO**

**Creatore:** bob@example.com (Affidabilità: 100%)

**Descrizione** [Modifica](#)

Mollit eiusmod deserunt sunt amet do eu anim ipsum sit. Fugiat consectetur aute duis eiusmod adipisicing quis laboris in. Consectetur voluptate cupidatat ipsum id elit.



**Durata:** 22/03/2025 - 31/12/2025    **283 GIORNI RIMASTI**

## Funzionalità principali:

- Finanziamento del progetto (se aperto)
- Visualizzazione delle reward disponibili

**Finanziamenti**  
Finanzia il progetto per aiutare il creatore a raggiungere il budget richiesto. Ogni finanziamento è ricompensato con una delle reward disponibili.

**Budget:** 10,000.00€

**38.84%**  
3,884.00€ / 10,000.00€

**Reward**  
Visualizza le reward disponibili per il progetto. Ogni reward è ottenibile con un finanziamento di un certo importo.

RWD_Default	RWD1_Alpha	RWD2_Alpha	RWD3_Alpha
Importo minimo: 0.01€ Reward di default 	Importo minimo: 150.00€ Do deserunt ullamco aliquip ad consequat Lorem minim irure. 	Importo minimo: 300.00€ Occaecat do elit consequat esse voluptate cillum fugiat. 	Importo minimo: 500.00€ Aliquip ea duis excepteur dolor elit prident ipsum qui. 

**Finanza il Progetto (€)**  
Inserisci l'importo che desideri finanziare e premi invia.

150.00

Invia

- Gestione delle componenti (per progetti hardware)
- Gestione dei profili e candidature (per progetti software)
- Sezione commenti con possibilità di risposta per il creatore

**Profilo**  
(Scorri per visualizzare i restanti) Seleziona un profilo per candidarti al progetto software. Assicurati di avere le competenze e il livello richiesto (X/5).

Backend Developer	Designer	Documentation Specialist	Frontend Developer
<ul style="list-style-type: none"> <li>• ERD (4/5)</li> <li>• MongoDB (4/5)</li> <li>• MySQL (4/5)</li> <li>• PHP (3/5)</li> </ul>	<ul style="list-style-type: none"> <li>• CSS (4/5)</li> <li>• Figma (4/5)</li> </ul>	<ul style="list-style-type: none"> <li>• ERD (4/5)</li> <li>• HTML (3/5)</li> <li>• Markdown (3/5)</li> <li>• UML (5/5)</li> </ul>	<ul style="list-style-type: none"> <li>• CSS (3/5)</li> <li>• HTML (3/5)</li> <li>• JavaScript (4/5)</li> </ul>

**Commenti**  
Lascia un commento per esprimere la tua opinione sul progetto.

**heidi\_90** 22/03/2025 18:48  
Amo il concetto di Alpha!

**Risposta (You)**  
Grazie per il supporto, Heidi!

**Elimina**

**mike89** 22/03/2025 18:48  
Gran lavoro, Bob!

**Risposta (You)**  
Grazie, Mike!

**Elimina**

**Commento**  
Inserisci un commento per esprimere la tua opinione sul progetto.

Invia

I creatori del progetto visualizzano anche bottoni di modifica per ogni sezione, che reindirizzano alle pagine di aggiornamento specifiche.

## CANDIDATURE

- `candidature.php`

La pagina candidature mostra tutte le candidature inviate dall'utente per partecipare ai progetti software. Per gli utenti creatori, è presente anche una sezione che visualizza le candidature ricevute dai propri progetti.

The screenshot shows a web application interface for managing project applications. At the top, there's a navigation bar with the brand name 'BOSTARTER' and links for 'Progetti', 'Finanziamenti', 'Statistiche', 'Curriculum', and 'Candidature'. On the right, it shows a 'Logout' link and an email address. Below the navigation is a section titled 'Candidature' with a yellow header bar labeled 'Candidature Inviate'. This section contains a table with one row for 'ProgettoAlpha'. The row includes columns for 'Progetto' (link to 'ProgettoAlpha'), 'Creatore' (bobby85), 'Profilo' (Frontend Developer), and 'Stato' (yellow button labeled 'In attesa'). A small note below the project name says: 'Mollit eiusmod deserunt sunt amet do eu anim ipsum sit. Fugiat consectetur aute duis eiusmod adipisicing quis laboris in. Consectetur voluptate cupidatat ipsum id elit.' At the bottom of the page, a dark footer bar displays the copyright notice: '© 2025 BOSTARTER. All rights reserved.'

Le candidature possono avere tre stati: "in attesa", "accettato" o "rifiutato". I creatori possono gestire le candidature ricevute, accettandole o rifiutandole. Un utente può candidarsi a un profilo solo se possiede tutte le competenze richieste con livello uguale o superiore a quello richiesto.

This screenshot shows the 'Candidature Ricevute' (Received Applications) section. It features a blue header bar labeled 'Candidature Ricevute'. Below it is a table with five columns: 'Progetto', 'Profilo', 'Candidato', 'Stato', and 'Azioni'. There are three rows of data for 'ProgettoAlpha': 1. Designer (mike89, mike@example.com) with status 'Accettato' and a neutral action button. 2. Project Manager (judge\_judy, judy@example.com) with status 'Accettato' and a neutral action button. 3. Frontend Developer (edo\_unibo, edoardo.galli3@studio.unibo.it) with status 'In attesa' and two buttons: 'Accetta' (green) and 'Rifiuta' (red). The table has a light gray background with alternating row colors.

## GESTIONE DESCRIZIONE/FOTO

- `progetto_aggiorna.php`

La pagina di gestione descrizione consente ai creatori di aggiornare il contenuto della descrizione del progetto, e di inserire/eliminare foto del progetto.

Torna al Progetto

### Aggiorna Descrizione

Nuova Descrizione

Sint consequat officia anim eu voluptate Lorem. velit sint labore excepteur laboris magna do esse. Irure ut aliquip sit aliquip esse qui proident culpa fugiat est elit veniam deserunt.

Aggiorna

### Inserisci/Elimina Foto

Elimina  Elimina  Elimina  Elimina 

Selezione Foto (Max 4MB)  
Inserisci una foto per il progetto.

Choose File no file selected

Inserisci Foto

## GESTIONE BUDGET

- progetto\_aggiorna.php

La pagina di gestione del budget consente ai creatori di aggiornare il budget richiesto per poter completare il progetto.

Torna al Progetto

### Aggiorna Budget

Budget Attuale: 12,850.00€  
Costo Componenti: 12,850.00€

43.19% Finanziato  


Nuovo Budget (€)  
Il nuovo budget non può essere inferiore al costo delle componenti (12,850.00€)

12850.00

Aggiorna Budget

Se il budget viene aggiornato ad un valore inferiore alla somma dei finanziamenti ricevuti, viene chiuso automaticamente. Ciò vale per entrambi tipi di progetto. Inoltre,

per i progetti di tipo hardware, il budget non può essere inferiore alla somma del costo delle componenti richieste nel progetto.

## GESTIONE REWARD

- `progetto_aggiorna.php`

La pagina di gestione reward consente ai creatori di inserire nuove reward per il progetto, oppure di aggiornare reward esistenti. Ogni progetto viene popolato con una reward di default.

The screenshot shows the 'Gestione Reward' interface. At the top, there's a section titled 'Reward Esistenti' displaying four reward cards:

- RWD\_Default**: Importo minimo: 0.01€, Reward di default. It features an icon of a blue gift box with a red ribbon.
- RWD1\_Beta**: Importo minimo: 150.00€. Description: Quis Lorem sit tempor ex eiusmod aliqua sint aliqua incididunt laborum occaecat.
- RWD2\_Beta**: Importo minimo: 300.00€. Description: Aute molit laboris veniam laborum id sunt magna consequat enim adipsicing.
- RWD3\_Beta**: Importo minimo: 500.00€. Description: Magna tempor in amet sit incididunt magna elit nostrud laborum in.

Below this is a section titled 'Aggiungi Nuova Reward' with fields for 'Codice Reward' (with placeholder 'Inserisci un codice univoco che identifichi questa reward.'), 'Importo Minimo (€)' (with placeholder 'Importo minimo per ottenere questa reward.' and value '0.01'), 'Descrizione Reward' (with placeholder 'Descrivi cosa riceverà l'utente con questa reward.'), and 'Foto Reward' (with placeholder 'Carica un'immagine per rappresentare questa reward (Max 4MB.)' and a 'Choose File' button).

A blue 'Inserisci Reward' button is located at the bottom of the form.

## GESTIONE PROFILI/COMPONENTI

- `progetto_aggiorna.php`

La pagina di gestione profili/componenti consente ai creatori di gestire i dettagli specifici del proprio progetto.

### Per progetti software:

- Creazione, modifica ed eliminazione di profili richiesti
- Definizione delle competenze necessarie per ciascun profilo e relativo livello richiesto

[Torna al Progetto](#)**Profili Esistenti**Backend Developer [Modifica](#) [Elimina](#)Competenze: [ERD \(4\)](#) [MongoDB \(4\)](#) [MySQL \(4\)](#) [PHP \(3\)](#)Designer [Modifica](#) [Elimina](#)Competenze: [CSS \(4\)](#) [Figma \(4\)](#)Documentation Specialist [Modifica](#) [Elimina](#)Competenze: [ERD \(4\)](#) [HTML \(3\)](#) [Markdown \(3\)](#) [UML \(5\)](#)Frontend Developer [Modifica](#) [Elimina](#)Competenze: [CSS \(3\)](#) [HTML \(3\)](#) [JavaScript \(4\)](#)Project Manager [Modifica](#) [Elimina](#)**Nuovo Profilo**

Nome Profilo

Es. API Developer

[Crea Profilo](#)

© 2025 BOSTARTER. All rights reserved.

Eventuali modifiche ai requisiti di un profilo possono comportare il rifiuto automatico di candidature esistenti se i candidati non soddisfano più i nuovi requisiti.

[Torna al Progetto](#)**Profili Esistenti** [Nuovo Profilo](#)Backend Developer [Modifica](#) [Elimina](#)Competenze: [ERD \(4\)](#) [MongoDB \(4\)](#) [MySQL \(4\)](#) [PHP \(3\)](#)Designer [Modifica](#) [Elimina](#)Competenze: [CSS \(4\)](#) [Figma \(4\)](#)Documentation Specialist [Modifica](#) [Elimina](#)Competenze: [ERD \(4\)](#) [HTML \(3\)](#) [Markdown \(3\)](#) [UML \(5\)](#)Frontend Developer [Modifica](#) [Elimina](#)Competenze: [CSS \(3\)](#) [HTML \(3\)](#) [JavaScript \(4\)](#)Project Manager [Modifica](#) [Elimina](#)**Modifica: Backend Developer**

Nome Profilo

Backend Developer

[Aggiorna Nome](#)**Competenze**

Competenza	Livello	Azioni
ERD	4	<a href="#">Modifica</a> <a href="#">Elimina</a>
MongoDB	4	<a href="#">Modifica</a> <a href="#">Elimina</a>
MySQL	4	<a href="#">Modifica</a> <a href="#">Elimina</a>
PHP	3	<a href="#">Modifica</a> <a href="#">Elimina</a>

**Aggiungi una competenza**

Competenza

Livello Richiesto (0-5)

Seleziona una competenza

3

[Aggiungi](#)

© 2025 BOSTARTER. All rights reserved.

**Per progetti hardware:**

- Aggiunta, modifica ed eliminazione di componenti necessari
- Definizione di quantità e prezzo di ciascun componente

[Torna al Progetto](#)**Componenti Esistenti**

Circuiti stampati    [Modifica](#)    [Elimina](#)  
 Descrizione: Tempor veniam ut culpa commodo ipsum do laboris laborum minim elit.  
 Quantità: 10   Prezzo: 150,00€   Totale: 1,500,00€

LED RGB    [Modifica](#)    [Elimina](#)  
 Descrizione: Fugiat et quis labore occaecat qui id do paratur culpa.  
 Quantità: 50   Prezzo: 1,00€   Totale: 50,00€

Metallo lavorato    [Modifica](#)    [Elimina](#)  
 Descrizione: Cupidatat excepteur ea aute commodo amet qui.  
 Quantità: 30   Prezzo: 200,00€   Totale: 6,000,00€

Sensori di movimento    [Modifica](#)    [Elimina](#)  
 Descrizione: Ullamco dolore nisi dolor paratur tempor paratur esse quis.  
 Quantità: 10   Prezzo: 350,00€   Totale: 5,250,00€

Viti    [Modifica](#)    [Elimina](#)  
 Descrizione: Ut eu ullamco quis volutate est incididunt duis eiusmod culpa et.  
 Quantità: 100   Prezzo: 0,50€   Totale: 50,00€

**Nuovo Componente**

**Informazioni Budget**  
 Budget attuale del progetto: **12,850,00€**  
 Costo totale attuale dei componenti: **12,850,00€**

**Nota:** Se il costo totale dei componenti supera il budget attuale del progetto, il budget verrà automaticamente incrementato per coprire il costo totale.

**Nome Componente**

Es. Scheda Arduino

**Descrizione**

Descrizione del componente...

**Quantità**

Es. 5

**Prezzo Unitario (€)**

Es. 25,99

[Crea Componente](#)

© 2025 BOSTARTER. All rights reserved.

Eventuali modifiche alla quantità o prezzo di componenti esistenti, o inserimento di nuovi, possono comportare l'aumento automatico del budget del progetto, se la somma del costo dei componenti eccede il budget attuale.

[Torna al Progetto](#)**Componenti Esistenti****Nuovo Componente**

Circuiti stampati    [Modifica](#)    [Elimina](#)  
 Descrizione: Tempor veniam ut culpa commodo ipsum do laboris laborum minim elit.  
 Quantità: 10   Prezzo: 150,00€   Totale: 1,500,00€

LED RGB    [Modifica](#)    [Elimina](#)  
 Descrizione: Fugiat et quis labore occaecat qui id do paratur culpa.  
 Quantità: 50   Prezzo: 1,00€   Totale: 50,00€

Metallo lavorato    [Modifica](#)    [Elimina](#)  
 Descrizione: Cupidatat excepteur ea aute commodo amet qui.  
 Quantità: 30   Prezzo: 200,00€   Totale: 6,000,00€

Sensori di movimento    [Modifica](#)    [Elimina](#)  
 Descrizione: Ullamco dolore nisi dolor paratur tempor paratur esse quis.  
 Quantità: 10   Prezzo: 350,00€   Totale: 5,250,00€

Viti    [Modifica](#)    [Elimina](#)  
 Descrizione: Ut eu ullamco quis volutate est incididunt duis eiusmod culpa et.  
 Quantità: 100   Prezzo: 0,50€   Totale: 50,00€

**Modifica: Circuiti stampati**

**Informazioni Budget**  
 Budget attuale del progetto: **12,850,00€**  
 Costo totale attuale dei componenti: **11,350,00€**

Costo attuale di questo componente: **1,500,00€**

**Nota:** Se il costo totale dei componenti supera il budget attuale del progetto, il budget verrà automaticamente incrementato per coprire il costo totale.

**Nome Componente**

Circuiti stampati

**Descrizione**

Tempor veniam ut culpa commodo ipsum do laboris laborum minim elit.

**Quantità**

10

**Prezzo Unitario (€)**

150,00

[Annulla](#)[Procedi](#)

© 2025 BOSTARTER. All rights reserved.

# LOGS

- logs.php

Gli utenti con privilegi di amministratore hanno accesso ad una pagina aggiuntive per la visualizzazione dei log di sistema, reperiti da MongoDB.

La pagina logs permette agli amministratori di monitorare tutte le attività sulla piattaforma, visualizzando le operazioni effettuate dagli utenti, suddivise per collezioni MongoDB corrispondenti alle tabelle del database.

The screenshot shows the BOSTARTER application interface. At the top, there is a navigation bar with links for Progetti, Finanziamenti, Statistiche, Curriculum, Candidature, and Logs. On the right side of the bar, it says "Logout (alice@example.com)". Below the navigation bar, the word "Logs" is centered. Underneath "Logs", there is a search bar labeled "Filia Log" and a checkbox labeled "Mostra solo errori". Below these are four buttons: "Tutte le collezioni" (highlighted in blue), "SKILL\_CURRICULUM", "UTENTE", and "FINANZIAMENTO". The main content area is titled "Tutti i Log" and displays a table of logs. The table has columns: Timestamp, Tabella, Azione, Utente, Stato, and Dettagli. There are 25 logs listed, all of which are "Success". Each log entry includes a "Dettagli" button. The timestamp for most logs is 2025-03-22 20:47:17, except for the last one which is 2025-03-22 19:42:57. The "Tabella" column shows "UTENTE" for most logs and "FINANZIAMENTO" for the last one. The "Azione" column shows "LOGIN" for most logs and "LOGOUT" for the last one. The "Utente" column shows various email addresses like alice@example.com, diana@example.com, bob@example.com, edoardo.galli3@studio.unibo.it, and edoardo.galli3@studio.unibo.it.

## Logs

Tutti i Log					
Timestamp	Tabella	Azione	Utente	Stato	Dettagli
2025-03-22 20:47:17	UTENTE	LOGIN	alice@example.com	Success	<button>Dettagli</button>
2025-03-22 20:47:16	UTENTE	LOGOUT	diana@example.com	Success	<button>Dettagli</button>
2025-03-22 20:37:58	UTENTE	LOGIN	diana@example.com	Success	<button>Dettagli</button>
2025-03-22 20:37:58	UTENTE	LOGOUT	bob@example.com	Success	<button>Dettagli</button>
2025-03-22 20:36:46	UTENTE	LOGIN	bob@example.com	Success	<button>Dettagli</button>
2025-03-22 20:36:44	UTENTE	LOGOUT	edoardo.galli3@studio.unibo.it	Success	<button>Dettagli</button>
2025-03-22 20:36:06	UTENTE	LOGIN	edoardo.galli3@studio.unibo.it	Success	<button>Dettagli</button>
2025-03-22 20:36:05	UTENTE	LOGOUT	bob@example.com	Success	<button>Dettagli</button>
2025-03-22 19:42:57	UTENTE	LOGIN	bob@example.com	Success	<button>Dettagli</button>

Ciascun log dispone di un modal che offre ulteriori dettagli sull'evento, come il file php che ha scatenato l'evento, il nome della stored procedure invocata, il feedback dal database/piattaforma, ecc.

The screenshot shows the BOSTARTER application interface with a modal window open. The modal is titled "Dettagli Log" and displays a table of log details. The table has columns: Timestamp, Tabella, Azione, Utente, Fonte, Procedura, Messaggio, and Stato. The timestamp is 2025-03-22 18:49:24. The tabella is FINANZIAMENTO, azione is INSERT, utente is edoardo.galli3@studio.unibo.it, fonte is finanziamento\_insert.php, procedura is sp\_finanziamento\_insert, messaggio is Finanziamento effettuato con successo, and stato is Success. To the right of the table, there is a sidebar with a "Mostra tutte le collezioni" button and a "FINANZIAMENTO" button. Below the table, there is a section titled "Data" with a JSON object:

```
{
  "p_email": "edoardo.galli3@studio.unibo.it",
  "p_nome_progetto": "ProgettoKappa",
  "p_codice_reward": "RWD_Default",
  "p_importo": 1212
}
```

At the bottom of the modal, it says "2 log trovati". The background of the main page shows the "Logs" section with a "Log - FINANZIAMENTO" heading and a table of logs. The first log in the table is for the same timestamp and action as the modal, showing a "Success" status. The footer of the page says "© 2025 BOSTARTER. All rights reserved."

## 5.3. LOGGING (MongoDB)

### Overview

La piattaforma implementa un sistema di logging centralizzato basato su MongoDB che traccia tutte le operazioni degli utenti. L'architettura di logging è encapsulata nella classe `EventPipeline`, che gestisce in modo uniforme la validazione, l'esecuzione delle operazioni sul database e la registrazione degli eventi.

Ogni azione è monitorata attraverso due scenari principali:

- **Fallimento:** Quando una validazione fallisce o si verifica un'eccezione, la pipeline invoca internamente la funzione `EventPipeline.fail()` che registra l'evento negativo in MongoDB e gestisce il redirect
- **Successo:** Al completamento dell'operazione, il metodo `EventPipeline.continue()` invoca la funzione `success()` che registra l'evento positivo e gestisce il redirect appropriato

Questo pattern garantisce che ogni interazione rilevante sia documentata con informazioni dettagliate (collezione, azione, procedura, utente, parametri) e sia facilmente consultabile dagli amministratori attraverso l'interfaccia dedicata.

Solo eventi significativi, ed eventi di fallimento di azioni/fetching di data dal database vengono registrate nel log, questo per evitare il sovraccarico di log inutili (es. utente non autenticato che cerca di accedere ad una pagina che richiede di essere autenticati) che rischiano di offuscare log più importanti.

### Accesso

I log sono disponibili a tutti gli utenti amministratori della piattaforma, e possono essere visualizzati nell'apposita pagina dei logs (`public/logs.php`). Se si desidera accedere esternamente ai database (MySQL e MongoDB) presenti all'interno del container, è sufficiente eseguire uno dei seguenti comandi sul terminale:

```
# MySQL
docker exec -it bostarter-db-1 mysql -uroot -ppassword BOSTARTER;

# MongoDB
docker exec -it bostarter-mongodb-1 mongosh BOSTARTER_LOG;
```

# 6. APPENDICE

---

## 6.1. SCRIPT

Di seguito viene illustrato il funzionamento dello script di inizializzazione della piattaforma. Assicurarsi di avergli dato il permesso di esecuzione, e di trovarsi alla radice del progetto quando lo si esegue:

```
#!/bin/bash
set -e

docker-compose down -v
export COMPOSE_BAKE=true
docker-compose up --build
```

- Viene eseguito `docker-compose down -v` per arrestare e rimuovere eventuali container e volumi esistenti.
- Viene eseguito `docker-compose up --build` per ricostruire e avviare i container.

Se si vede comparire sul terminale il seguente log, allora l'inizializzazione della piattaforma è andata a buon fine:

```
web-1      | === SEEDING seed_data.php START! ===
web-1      | Seeding ProgettoAlpha... OK.
web-1      | Seeding ProgettoBeta... OK.
web-1      | Seeding remaining projects... OK.
web-1      | === SEEDING seed_data.php COMPLETE! ===
web-1      |
web-1      | === BOSTARTER INIZIALIZZATO. PIATTAFORMA PRONTA! ===
```

Questo script automatizza l'intero processo di configurazione, garantendo che tutte le dipendenze siano installate e che l'ambiente Docker sia correttamente ricostruito, rendendo l'avvio della piattaforma semplice e veloce.

## 6.2. SCHEMA

Di seguito viene riportato il codice SQL completo utilizzato per la generazione della base di dati **BOSTARTER** e delle relative funzionalità (procedure, viste, trigger, eventi):

```

-- =====
--          BOSTARTER INIT
-- =====

DROP DATABASE IF EXISTS BOSTARTER;
CREATE DATABASE BOSTARTER CHARACTER SET utf8mb4 COLLATE
utf8mb4_unicode_ci;;
USE BOSTARTER;

-- =====
-- TABELLE
-- =====

-- 1. UTENTE
CREATE TABLE UTENTE (
    email            VARCHAR(100) NOT NULL,
    password         VARCHAR(255) NOT NULL,
    nickname         VARCHAR(50)  NOT NULL UNIQUE CHECK (
        LENGTH(nickname) > 0 ), -- Minimo 1 carattere
    nome             VARCHAR(50)  NOT NULL,
    cognome          VARCHAR(50)  NOT NULL,
    anno_nascita     INT        NOT NULL CHECK ( anno_nascita < 2007 ),
    -- età > 18
    luogo_nascita    VARCHAR(50)  NOT NULL,
    PRIMARY KEY (email)
) ENGINE = InnoDB
CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;

-- 2. ADMIN
CREATE TABLE ADMIN (
    email_utente      VARCHAR(100) NOT NULL,
    codice_sicurezza  VARCHAR(100) NOT NULL CHECK (
        LENGTH(codice_sicurezza) >= 8 ), -- Minimo 8 caratteri
    PRIMARY KEY (email_utente),
    CONSTRAINT fk_admin_utente
        FOREIGN KEY (email_utente)
            REFERENCES UTENTE (email)
            ON DELETE CASCADE
            ON UPDATE CASCADE
) ENGINE = InnoDB
CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;

-- 3. CREATORE
CREATE TABLE CREATORE (
    email_utente      VARCHAR(100) NOT NULL,
    nr_progetti      INT UNSIGNED DEFAULT 0,
    affidabilita     DECIMAL(5, 2) DEFAULT 0.00 CHECK ( affidabilita
    BETWEEN 0.00 AND 100.00 ), -- Progetti finanziati / Progetti creati
    PRIMARY KEY (email_utente),

```

```

CONSTRAINT fk_creatore_utente
    FOREIGN KEY (email_utente)
        REFERENCES UTENTE (email)
        ON DELETE CASCADE
        ON UPDATE CASCADE
) ENGINE = InnoDB
CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;

-- 4. PROGETTO
CREATE TABLE PROGETTO (
    nome          VARCHAR(100)  NOT NULL CHECK ( LENGTH(nome) > 0
),      -- Minimo 1 carattere
    email_creatore  VARCHAR(100)  NOT NULL,
    descrizione     TEXT        NOT NULL CHECK (
LENGTH(descrizione) > 0 ), -- Minimo 1 carattere
    budget         DECIMAL(10, 2) NOT NULL CHECK ( budget > 0 ),
    stato          ENUM ('aperto','chiuso') DEFAULT 'aperto',
    data_inserimento DATE      NOT NULL DEFAULT (CURRENT_DATE),
    data_limite     DATE      NOT NULL,
PRIMARY KEY (nome),
CONSTRAINT fk_progetto_creatore
    FOREIGN KEY (email_creatore)
        REFERENCES UTENTE (email)
        ON DELETE CASCADE
        ON UPDATE CASCADE
) ENGINE = InnoDB
CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;

-- 5. PROGETTO_SOFTWARE
CREATE TABLE PROGETTO_SOFTWARE (
    nome_progetto VARCHAR(100) NOT NULL,
PRIMARY KEY (nome_progetto),
CONSTRAINT fk_psw_progetto
    FOREIGN KEY (nome_progetto)
        REFERENCES PROGETTO (nome)
        ON DELETE CASCADE
        ON UPDATE CASCADE
) ENGINE = InnoDB
CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;

-- 6. PROGETTO_HARDWARE
CREATE TABLE PROGETTO_HARDWARE (
    nome_progetto VARCHAR(100) NOT NULL,
PRIMARY KEY (nome_progetto),
CONSTRAINT fk_phw_progetto
    FOREIGN KEY (nome_progetto)
        REFERENCES PROGETTO (nome)
        ON DELETE CASCADE
        ON UPDATE CASCADE
) ENGINE = InnoDB

```

```

CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;

-- 7. FOTO
CREATE TABLE FOTO (
    id          INT          NOT NULL AUTO_INCREMENT,
    nome_progetto VARCHAR(100) NOT NULL,
    foto        MEDIUMBLOB   NOT NULL, -- Max 16 MB per foto
    PRIMARY KEY (id, nome_progetto),
    CONSTRAINT fk_foto_progetto
        FOREIGN KEY (nome_progetto)
            REFERENCES PROGETTO (nome)
            ON DELETE CASCADE
            ON UPDATE CASCADE
) ENGINE = InnoDB
CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;

-- 8. REWARD
CREATE TABLE REWARD (
    codice      VARCHAR(50)  NOT NULL,
    nome_progetto VARCHAR(100) NOT NULL,
    descrizione TEXT        NOT NULL CHECK ( LENGTH(descrizione)
> 0 ), -- Minimo 1 carattere
    foto        MEDIUMBLOB  NOT NULL,
    min_importo DECIMAL(10, 2) NOT NULL CHECK ( min_importo > 0 ),
    PRIMARY KEY (codice, nome_progetto),
    CONSTRAINT fk_reward_progetto
        FOREIGN KEY (nome_progetto)
            REFERENCES PROGETTO (nome)
            ON DELETE CASCADE
            ON UPDATE CASCADE
) ENGINE = InnoDB
CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;

-- 9. COMPONENTE
CREATE TABLE COMPONENTE (
    nome_componente VARCHAR(100)  NOT NULL CHECK (
LENGTH(nome_componente) > 0 ), -- Minimo 1 carattere
    nome_progetto   VARCHAR(100)  NOT NULL,
    descrizione    TEXT         NOT NULL CHECK (
LENGTH(descrizione) > 0 ), -- Minimo 1 carattere
    quantita       INT          NOT NULL CHECK ( quantita > 0 ),
    -- Business Rule #3
    prezzo         DECIMAL(10, 2) NOT NULL CHECK ( prezzo > 0 ),
    PRIMARY KEY (nome_componente, nome_progetto),
    CONSTRAINT fk_comp_phw
        FOREIGN KEY (nome_progetto)
            REFERENCES PROGETTO_HARDWARE (nome_progetto)
            ON DELETE CASCADE
            ON UPDATE CASCADE
) ENGINE = InnoDB

```

```

CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;

-- 10. PROFILO
CREATE TABLE PROFILO (
    nome_profilo VARCHAR(100) NOT NULL CHECK ( LENGTH(nome_profilo) > 0 ), -- Minimo 1 carattere
    nome_progetto VARCHAR(100) NOT NULL,
    PRIMARY KEY (nome_profilo, nome_progetto),
    CONSTRAINT fk_profilo_progetto
        FOREIGN KEY (nome_progetto)
            REFERENCES PROGETTO (nome)
            ON DELETE CASCADE
            ON UPDATE CASCADE
) ENGINE = InnoDB
CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;

-- 11. SKILL
CREATE TABLE SKILL (
    competenza VARCHAR(100) NOT NULL CHECK ( LENGTH(competenza) > 0 ), -- Minimo 1 carattere
    PRIMARY KEY (competenza)
) ENGINE = InnoDB
CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;

-- 12. FINANZIAMENTO
CREATE TABLE FINANZIAMENTO (
    data DATE NOT NULL DEFAULT (CURRENT_DATE),
    email_utente VARCHAR(100) NOT NULL,
    nome_progetto VARCHAR(100) NOT NULL,
    codice_reward VARCHAR(50) NOT NULL,
    importo DECIMAL(10, 2) NOT NULL CHECK ( importo >= 0.01 ),
    PRIMARY KEY (data, email_utente, nome_progetto),
    CONSTRAINT fk_fin_utente
        FOREIGN KEY (email_utente)
            REFERENCES UTENTE (email)
            ON DELETE CASCADE
            ON UPDATE CASCADE,
    CONSTRAINT fk_fin_progetto
        FOREIGN KEY (nome_progetto)
            REFERENCES PROGETTO (nome)
            ON DELETE CASCADE
            ON UPDATE CASCADE,
    CONSTRAINT fk_fin_reward
        FOREIGN KEY (codice_reward, nome_progetto)
            REFERENCES REWARD (codice, nome_progetto)
            ON DELETE CASCADE
            ON UPDATE CASCADE
) ENGINE = InnoDB
CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;

```

```
-- 13. COMMENTO
CREATE TABLE COMMENTO (
    id          INT          NOT NULL AUTO_INCREMENT,
    email_utente VARCHAR(100) NOT NULL,
    nome_progetto VARCHAR(100) NOT NULL,
    data        DATETIME     NOT NULL DEFAULT CURRENT_TIMESTAMP,
    testo       TEXT         NOT NULL CHECK ( LENGTH(testo) > 0 ), -
    - Minimo 1 carattere
    risposta    TEXT         NULL,
    - Business Rule #8
    PRIMARY KEY (id),
    CONSTRAINT fk_com_utente
        FOREIGN KEY (email_utente)
            REFERENCES UTENTE (email)
            ON DELETE CASCADE
            ON UPDATE CASCADE,
    CONSTRAINT fk_com_progetto
        FOREIGN KEY (nome_progetto)
            REFERENCES PROGETTO (nome)
            ON DELETE CASCADE
            ON UPDATE CASCADE
) ENGINE = InnoDB
CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
```

```
-- 14. SKILL_CURRICULUM
CREATE TABLE SKILL_CURRICULUM (
    email_utente      VARCHAR(100) NOT NULL,
    competenza        VARCHAR(100) NOT NULL,
    livello_effettivo TINYINT      NOT NULL CHECK ( livello_effettivo
BETWEEN 0 AND 5 ), -- Business Rule #1
    PRIMARY KEY (email_utente, competenza),
    CONSTRAINT fk_skcurr_utente
        FOREIGN KEY (email_utente)
            REFERENCES UTENTE (email)
            ON DELETE CASCADE
            ON UPDATE CASCADE,
    CONSTRAINT fk_skcurr_skill
        FOREIGN KEY (competenza)
            REFERENCES SKILL (competenza)
            ON DELETE CASCADE
            ON UPDATE CASCADE
) ENGINE = InnoDB
CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
```

```
-- 15. SKILL_PROFILo
CREATE TABLE SKILL_PROFILo (
    nome_profilo      VARCHAR(100) NOT NULL,
    competenza        VARCHAR(100) NOT NULL,
    nome_progetto     VARCHAR(100) NOT NULL,
    livello_richiesto TINYINT      NOT NULL CHECK ( livello_richiesto
```

```

BETWEEN 0 AND 5 ), -- Business Rule #2
    PRIMARY KEY (nome_profilo, competenza, nome_progetto),
    CONSTRAINT fk_skprof_profilo
        FOREIGN KEY (nome_profilo, nome_progetto)
            REFERENCES PROFILO (nome_profilo, nome_progetto)
            ON DELETE CASCADE
            ON UPDATE CASCADE,
    CONSTRAINT fk_skprof_skill
        FOREIGN KEY (competenza)
            REFERENCES SKILL (competenza)
            ON DELETE CASCADE
            ON UPDATE CASCADE,
    CONSTRAINT fk_skprof_progetto
        FOREIGN KEY (nome_progetto)
            REFERENCES PROGETTO (nome)
            ON DELETE CASCADE
            ON UPDATE CASCADE
) ENGINE = InnoDB
CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;

-- 16. PARTECIPANTE
CREATE TABLE PARTECIPANTE (
    email_utente VARCHAR(100) NOT NULL,
    nome_progetto VARCHAR(100) NOT NULL,
    nome_profilo VARCHAR(100) NOT NULL,
    stato ENUM ('accettato','rifiutato','potenziale') DEFAULT
    'potenziale',
    PRIMARY KEY (email_utente, nome_progetto, nome_profilo),
    CONSTRAINT fk_part_utente
        FOREIGN KEY (email_utente)
            REFERENCES UTENTE (email)
            ON DELETE CASCADE
            ON UPDATE CASCADE,
    CONSTRAINT fk_part_progetto
        FOREIGN KEY (nome_progetto)
            REFERENCES PROGETTO (nome)
            ON DELETE CASCADE
            ON UPDATE CASCADE,
    CONSTRAINT fk_part_profilo
        FOREIGN KEY (nome_profilo, nome_progetto)
            REFERENCES PROFILO (nome_profilo, nome_progetto)
            ON DELETE CASCADE
            ON UPDATE CASCADE
) ENGINE = InnoDB
CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;

-- =====
-- STORED PROCEDURES (HELPER)
-- =====

```

```

-- In questo blocco vengono definite le stored procedure di controllo
("helper") che verranno utilizzate dalle stored procedure principali
("main").
-- Sono inclusi qui solo i controlli che sono comuni a più stored
procedure, per evitare duplicazione di codice. Laddove un controllo
sia specifico
-- per una sola stored procedure, esso è incluso direttamente
all'interno di quella stored procedure principale, e non qui.

-- SINTASSI GENERALE: sp_nome_procedura_check

-- L'unica eccezione sono le GENERICHE/UTILS sp_util_* che contengono
controlli generici che possono essere utilizzati da più stored
procedure principali e helper.

-- Le stored procedure che fanno SQLSTATE SIGNAL sono utilizzate
all'interno delle stored procedure principali per lanciare un errore
specifico.

-- Le stored procedure che restituiscono TRUE/FALSE sono
utilizzate al livello di applicazione/php per fare controlli
condizionali.

DELIMITER //

-- GENERICHE/UTILS: Insieme di controlli generici per tutte le stored
procedure principali laddove necessario

/*
* PROCEDURE: sp_util_utente_is_admin
* PURPOSE: Verifica se l'utente è un admin, lanciando un errore se
non lo è.
*
* @param IN p_email - Email dell'utente da controllare
*
* @throws 45000 - UTENTE NON ADMIN
*/
CREATE PROCEDURE sp_util_utente_is_admin(
    IN p_email VARCHAR(100)
)
BEGIN
    IF NOT EXISTS (SELECT 1
                    FROM ADMIN
                    WHERE email_utente = p_email) THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Utente non e\' un admin.';
    END IF;
END//


/*
* PROCEDURE: sp_util_utente_is_creatore
* PURPOSE: Verifica se l'utente è un creatore.

```

```

/*
*  @param IN p_email - Email dell'utente da controllare
*
*  @throws 45000 - UTENTE NON CREATORE
*/
CREATE PROCEDURE sp_util_utente_is_creatore(
    IN p_email VARCHAR(100)
)
BEGIN
    IF NOT EXISTS (SELECT 1
                    FROM CREATORE
                    WHERE email_utente = p_email) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Utente non e\' un creatore.';
    END IF;
END//


/*
*  PROCEDURE: sp_util_creatore_is_progetto_owner
*  PURPOSE: Verifica se l'utente è il creatore del progetto.
*
*  @param IN p_email - Email dell'utente da controllare
*  @param IN p_nome_progetto - Nome del progetto da controllare
*
*  @throws 45000 - UTENTE NON CREATORE DEL PROGETTO
*/
CREATE PROCEDURE sp_util_creatore_is_progetto_owner(
    IN p_email VARCHAR(100),
    IN p_nome_progetto VARCHAR(100)
)
BEGIN
    IF NOT EXISTS (SELECT 1
                    FROM PROGETTO
                    WHERE nome = p_nome_progetto
                    AND email_creatore = p_email) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Utente non e\' il creatore del
progetto.';
    END IF;
END//


/*
*  PROCEDURE: sp_util_reward_valid_finanziamento
*  PURPOSE: Verifica se una specifica reward è valida per un
determinato importo di finanziamento.
*          Se la reward non esiste o l'importo è insufficiente,
genera un errore.
*
*  @param IN p_nome_progetto - Nome del progetto
*  @param IN p_codice_reward - Codice della reward da verificare

```

```

/*
*   @param IN p_importo - Importo del finanziamento
*
*   @throws 45000 - REWARD NON TROVATA
*   @throws 45000 - IMPORTO INSUFFICIENTE PER LA REWARD
*/
CREATE PROCEDURE sp_util_reward_valid_finanziamento(
    IN p_nome_progetto VARCHAR(100),
    IN p_codice_reward VARCHAR(50),
    IN p_importo DECIMAL(10, 2)
)
BEGIN
    DECLARE min_importo_required DECIMAL(10, 2);

    -- Verifica che la reward esista per il progetto
    SELECT min_importo
    INTO min_importo_required
    FROM REWARD
    WHERE nome_progetto = p_nome_progetto
        AND codice = p_codice_reward;

    IF min_importo_required IS NULL THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Reward non trovata.';
    END IF;

    -- Verifica che l'importo sia sufficiente per la reward
    IF p_importo < min_importo_required THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Importo insufficiente per la reward
selezionata.';
    END IF;
END//


/*
*   PROCEDURE: sp_util_progetto_exists
*   PURPOSE: Verifica se il progetto esiste.
*
*   @param IN p_nome_progetto - Nome del progetto da controllare
*
*   @throws 45000 - PROGETTO NON ESISTENTE
*/
CREATE PROCEDURE sp_util_progetto_exists(
    IN p_nome_progetto VARCHAR(100)
)
BEGIN
    IF NOT EXISTS (SELECT 1
                    FROM PROGETTO
                    WHERE nome = p_nome_progetto) THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Il progetto non esiste.';
```

```

        END IF;
    END//


/*
*  PROCEDURE: sp_util_progetto_is_software
*  PURPOSE: Verifica se il progetto è di tipo software.
*
*  @param IN p_nome_progetto - Nome del progetto da controllare
*
*  @throws 45000 - PROGETTO NON DI TIPO SOFTWARE
*/
CREATE PROCEDURE sp_util_progetto_is_software(
    IN p_nome_progetto VARCHAR(100)
)
BEGIN
    IF NOT EXISTS (SELECT 1
                    FROM PROGETTO_SOFTWARE
                    WHERE nome_progetto = p_nome_progetto) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Il progetto non e\' di tipo
software.';
    END IF;
END//


/*
*  PROCEDURE: sp_util_progetto_is_hardware
*  PURPOSE: Verifica se il progetto è di tipo hardware.
*
*  @param IN p_nome_progetto - Nome del progetto da controllare
*
*  @throws 45000 - PROGETTO NON DI TIPO HARDWARE
*/
CREATE PROCEDURE sp_util_progetto_is_hardware(
    IN p_nome_progetto VARCHAR(100)
)
BEGIN
    IF NOT EXISTS (SELECT 1
                    FROM PROGETTO_HARDWARE
                    WHERE nome_progetto = p_nome_progetto) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Il progetto non e\' di tipo
hardware.';
    END IF;
END//


/*
*  PROCEDURE: sp_util_profilo_exists
*  PURPOSE: Verifica se il profilo di un progetto esiste.
*
*  @param IN p_nome_profilo - Nome del profilo da controllare

```

```

*  @param IN p_nome_progetto - Nome del progetto a cui appartiene il
profilo
*
*  @throws 45000 - PROFILO NON ESISTENTE
*/
CREATE PROCEDURE sp_util_profilo_exists(
    IN p_nome_profilo VARCHAR(100),
    IN p_nome_progetto VARCHAR(100)
)
BEGIN
    IF NOT EXISTS (SELECT 1
                    FROM PROFILO
                    WHERE nome_profilo = p_nome_profilo
                        AND nome_progetto = p_nome_progetto) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Profilo non esistente.';
    END IF;
END//


/*
*  PROCEDURE: sp_util_skill_profilo_exists
*  PURPOSE: Verifica se la competenza richiesta dal profilo di un
progetto esiste.
*
*  @param IN p_nome_profilo - Nome del profilo da controllare
*  @param IN p_nome_progetto - Nome del progetto a cui appartiene il
profilo
*  @param IN p_competenza - Competenza richiesta dal profilo
*
*  @throws 45000 - COMPETENZA NON PRESENTE NEL PROFILO
*/
CREATE PROCEDURE sp_util_skill_profilo_exists(
    IN p_nome_profilo VARCHAR(100),
    IN p_nome_progetto VARCHAR(100),
    IN p_competenza VARCHAR(100)
)
BEGIN
    IF NOT EXISTS (SELECT 1
                    FROM SKILL_PROFIL0
                    WHERE nome_profilo = p_nome_profilo
                        AND nome_progetto = p_nome_progetto
                        AND competenza = p_competenza) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Competenza non presente nel profilo.';
    END IF;
END//


/*
*  PROCEDURE: sp_util_commento_exists
*  PURPOSE: Verifica se il commento esiste

```

```

/*
*  @param IN p_id - ID del commento da controllare
*
*  @throws 45000 - COMMENTO NON ESISTENTE
*/
CREATE PROCEDURE sp_util_commento_exists(
    IN p_id INT
)
BEGIN
    IF NOT EXISTS (SELECT 1
                    FROM COMMENTO
                    WHERE id = p_id) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Il commento non esiste.';
    END IF;
END//


/*
*  PROCEDURE: sp_util_admin_exists
*  PURPOSE: Verifica se l'utente è un admin, selezionando TRUE se lo
è, FALSE altrimenti.
*
*  @param IN p_email - Email dell'utente da controllare
*/
CREATE PROCEDURE sp_util_admin_exists(
    IN p_email VARCHAR(100)
)
BEGIN
    START TRANSACTION;
    IF EXISTS (SELECT 1 FROM ADMIN WHERE email_utente = p_email) THEN
        SELECT TRUE AS is_admin;
    ELSE
        SELECT FALSE AS is_admin;
    END IF;
    COMMIT;
END//


/*
*  PROCEDURE: sp_util_creatore_exists
*  PURPOSE: Verifica se l'utente è un creatore, selezionando TRUE se
lo è, FALSE altrimenti.
*
*  @param IN p_email - Email dell'utente da controllare
*/
CREATE PROCEDURE sp_util_creatore_exists(
    IN p_email VARCHAR(100)
)
BEGIN
    START TRANSACTION;
    IF EXISTS (SELECT 1 FROM CREATORE WHERE email_utente = p_email)

```

```

THEN
    SELECT TRUE AS is_creatore;
ELSE
    SELECT FALSE AS is_creatore;
END IF;
COMMIT;
END//


/*
* PROCEDURE: sp_util_progetto_owner_exists
* PURPOSE: Verifica se l'utente è il creatore del progetto,
selezionando TRUE se lo è, FALSE altrimenti.
*
* @param IN p_email - Email dell'utente da controllare
* @param IN p_nome_progetto - Nome del progetto da controllare
*/
CREATE PROCEDURE sp_util_progetto_owner_exists(
    IN p_email VARCHAR(100),
    IN p_nome_progetto VARCHAR(100)
)
BEGIN
    START TRANSACTION;
    IF EXISTS (SELECT 1 FROM PROGETTO WHERE email_creatore = p_email
AND nome = p_nome_progetto) THEN
        SELECT TRUE AS is_owner;
    ELSE
        SELECT FALSE AS is_owner;
    END IF;
    COMMIT;
END//


/*
* PROCEDURE: sp_util_progetto_type
* PURPOSE: Restituisce il tipo di progetto (software/hardware).
*
* @param IN p_nome_progetto - Nome del progetto da controllare
*/
CREATE PROCEDURE sp_util_progetto_type(
    IN p_nome_progetto VARCHAR(100)
)
BEGIN
    START TRANSACTION;
    IF EXISTS (SELECT 1 FROM PROGETTO_SOFTWARE WHERE nome_progetto =
p_nome_progetto) THEN
        SELECT 'SOFTWARE' AS tipo_progetto;
    ELSE
        SELECT 'HARDWARE' AS tipo_progetto;
    END IF;
    COMMIT;
END//

```

```

/*
*  PROCEDURE: sp_util_progetto_is_aperto
*  PURPOSE: Verifica se lo stato del progetto è aperto.
*
*  @param IN p_nome_progetto - Nome del progetto da controllare
*
*  @throws 45000 - PROGETTO CHIUSO
*/
CREATE PROCEDURE sp_util_progetto_is_aperto(
    IN p_nome_progetto VARCHAR(100)
)
BEGIN
    IF (SELECT stato FROM PROGETTO WHERE nome = p_nome_progetto) =
    'chiuso' THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Il progetto e\' chiuso.';
    END IF;
END//


/*
*  PROCEDURE: sp_util_admin_get_codice_sicurezza
*  PURPOSE: Recuperare il codice di sicurezza di un admin.
*  USED BY: ADMIN
*
*  @param IN p_email - Email dell'admin da controllare
*
*/
CREATE PROCEDURE sp_util_admin_get_codice_sicurezza(
    IN p_email VARCHAR(100)
)
BEGIN
    START TRANSACTION;
    SELECT codice_sicurezza
    FROM ADMIN
    WHERE email_utente = p_email;
    COMMIT;
END//


/*
*  PROCEDURE: sp_util_utente_convert_creatore
*  PURPOSE: Conversione di un utente in creatore.
*  USED BY: UTENTE
*
*  @param IN p_email - Email dell'utente
*
*  @throws 45000 - EMAIL NON VALIDA
*  @throws 45000 - UTENTE GIA' CREATORE
*/
CREATE PROCEDURE sp_util_utente_convert_creatore(

```

```

    IN p_email VARCHAR(100)
)
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN
            ROLLBACK;
            RESIGNAL;
        END;
    START TRANSACTION;
    -- Controllo che l'utente esista
    IF NOT EXISTS (SELECT 1 FROM UTENTE WHERE email = p_email) THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Email non valida.';
    END IF;

    -- Controllo che l'utente non sia già un creatore
    IF EXISTS (SELECT 1 FROM CREATORE WHERE email_utente = p_email)
THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Utente ha già il ruolo di creatore.';
    END IF;

    -- Conversione dell'utente in creatore
    INSERT INTO CREATORE (email_utente)
VALUES (p_email);
    COMMIT;
END//


/*
*  PROCEDURE: sp_util_creatore_get_affidabilita
*  PURPOSE: Visualizzazione dell'affidabilità di un creatore.
*  USED BY: CREATORE
*
*  @param IN p_email - Email del creatore
*/
CREATE PROCEDURE sp_util_creatore_get_affidabilita(
    IN p_email VARCHAR(100)
)
BEGIN
    START TRANSACTION;
    SELECT affidabilita
    FROM CREATORE
    WHERE email_utente = p_email;
    COMMIT;
END//


/*
*  PROCEDURE: sp_util_creatore_get_nr_progetti
*  PURPOSE: Visualizzazione del numero di progetti creati da un
creatore.

```

```

* USED BY: CREATORE
*
* @param IN p_email - Email del creatore
*/
CREATE PROCEDURE sp_util_creatore_get_nr_progetti(
    IN p_email VARCHAR(100)
)
BEGIN
    START TRANSACTION;
    SELECT nr_progetti
    FROM CREATORE
    WHERE email_utente = p_email;
    COMMIT;
END//


/*
* PROCEDURE: sp_util_creatore_get_tot_partecipanti
* PURPOSE: Calcola il numero totale di partecipanti accettati ai
progetti di un creatore.
* USED BY: CREATORE
*
* @param IN p_email - Email del creatore
*/
CREATE PROCEDURE sp_util_creatore_get_tot_partecipanti(
    IN p_email VARCHAR(100)
)
BEGIN
    START TRANSACTION;
    SELECT COUNT(*) AS total_partecipanti
    FROM PARTECIPANTE P
        JOIN PROGETTO PR ON P.nome_progetto = PR.nome
    WHERE PR.email_creatore = p_email
        AND P.stato = 'accettato';
    COMMIT;
END//


/*
* PROCEDURE: sp_util_utente_finanziato_progetto_oggi
* PURPOSE: Verifica se l'utente ha finanziato il progetto oggi.
Restituisce TRUE se ha finanziato, FALSE altrimenti.
* USED BY: ALL
*
* @param IN p_email - Email dell'utente
* @param IN p_nome_progetto - Nome del progetto
*/
CREATE PROCEDURE sp_util_utente_finanziato_progetto_oggi(
    IN p_email VARCHAR(100),
    IN p_nome_progetto VARCHAR(100)
)
BEGIN

```

```

START TRANSACTION;
IF EXISTS (SELECT 1
            FROM FINANZIAMENTO
            WHERE email_utente = p_email
              AND nome_progetto = p_nome_progetto
              AND data = CURRENT_DATE) THEN
    SELECT TRUE AS finanziato_oggi;
ELSE
    SELECT FALSE AS finanziato_oggi;
END IF;
COMMIT;
END//


/*
* PROCEDURE: sp_util_progetto_componenti_costo
* PURPOSE: Calcola il costo totale dei componenti di un progetto hardware.
* USED BY: CREATORE
*
* @param IN p_nome_progetto - Nome del progetto
*/
CREATE PROCEDURE sp_util_progetto_componenti_costo(
    IN p_nome_progetto VARCHAR(100)
)
BEGIN
    START TRANSACTION;
    SELECT IFNULL(SUM(prezzo * quantita), 0) AS costo_totale
    FROM COMPONENTE
    WHERE nome_progetto = p_nome_progetto;
    COMMIT;
END//


/*
* PROCEDURE: sp_util_partecipante_is_eligible
* PURPOSE: Verifica se un utente ha tutte le competenze necessarie per candidarsi a un profilo.
* USED BY: ALL
*
* @param IN p_email_utente - Email dell'utente
* @param IN p_nome_progetto - Nome del progetto
* @param IN p_nome_profilo - Nome del profilo
*/
CREATE PROCEDURE sp_util_partecipante_is_eligible(
    IN p_email_utente VARCHAR(100),
    IN p_nome_progetto VARCHAR(100),
    IN p_nome_profilo VARCHAR(100)
)
BEGIN
    DECLARE is_eligible BOOLEAN DEFAULT TRUE;
    DECLARE missing_skills INT;

```

```

START TRANSACTION;

    -- Conta quante competenze richieste dal profilo NON sono
    soddisfatte dall'utente
    SELECT COUNT(*)
    INTO missing_skills
    FROM SKILL_PROFIL0 sp
        LEFT JOIN SKILL_CURRICULUM sc ON sp.competenza =
    sc.competenza AND sc.email_utente = p_email_utente
        WHERE sp.nome_profilo = p_nome_profilo
            AND sp.nome_progetto = p_nome_progetto
            AND (sc.livello_effettivo IS NULL OR sc.livello_effettivo <
    sp.livello_richiesto);

    -- Se c'è almeno una competenza mancante o insufficiente, l'utente
    non è idoneo
    IF missing_skills > 0 THEN
        SET is_eligible = FALSE;
    END IF;

    -- Restituisce il risultato
    SELECT is_eligible AS eligible;
    COMMIT;
END//


-- SKILL_PROFIL0: sp_skill_profilo_check, USATO IN:
-- sp_skill_profilo_insert
-- sp_skill_profilo_delete
-- sp_skill_profilo_update

/*
*  PROCEDURE: sp_skill_profilo_check
*  PURPOSE: Controlla la validità di un profilo e di una competenza
richiesta.
*
*  @param IN p_nome_profilo - Nome del profilo da controllare
*  @param IN p_email_creatore - Email dell'utente creatore del
progetto
*  @param IN p_nome_progetto - Nome del progetto a cui appartiene il
profilo
*  @param IN p_competenza - Competenza richiesta dal profilo
*  @param IN p_is_insert - Flag per distinguere tra insert, update e
delete
*/
CREATE PROCEDURE sp_skill_profilo_check(
    IN p_nome_profilo VARCHAR(100),
    IN p_email_creatore VARCHAR(100),
    IN p_nome_progetto VARCHAR(100),
    IN p_competenza VARCHAR(100),

```

```

    IN p_is_insert BOOLEAN
)
BEGIN
    -- Controllo che il progetto esista
    CALL sp_util_progetto_exists(p_nome_progetto);

    -- Controllo che il progetto sia creato dall'utente
    CALL sp_util_creatore_is_progetto_owner(p_email_creatore,
p_nome_progetto);

    -- Controllo che il progetto sia di tipo software
    CALL sp_util_progetto_is_software(p_nome_progetto);

    -- Controllo che il profilo del progetto esista
    CALL sp_util_profilo_exists(p_nome_profilo, p_nome_progetto);

    -- Controllo che il profilo abbia la competenza richiesta (solo
per update e delete)
    IF NOT p_is_insert THEN
        CALL sp_util_skill_profilo_exists(p_nome_profilo,
p_nome_progetto, p_competenza);
    END IF;
END//


-- PROFILO: sp_profilo_check, USATO IN:
-- sp_profilo_insert
-- sp_profilo_delete
-- sp_profilo_nome_update

/*
*  PROCEDURE: sp_profilo_check
*  PURPOSE: Controlla la validità di un profilo.
*
*  @param IN p_nome_profilo - Nome del profilo da controllare
*  @param IN p_email_creatore - Email dell'utente creatore del
profilo
*  @param IN p_nome_progetto - Nome del progetto a cui appartiene il
profilo
*  @param IN p_is_insert - Flag per distinguere tra insert e delete
*/
CREATE PROCEDURE sp_profilo_check(
    IN p_nome_profilo VARCHAR(100),
    IN p_email_creatore VARCHAR(100),
    IN p_nome_progetto VARCHAR(100),
    IN p_is_insert BOOLEAN
)
BEGIN
    -- Controllo che il progetto esista
    CALL sp_util_progetto_exists(p_nome_progetto);

```

```

-- Controllo che il progetto sia creato dall'utente
CALL sp_util_creatore_is_progetto_owner(p_email_creatore,
p_nome_progetto);

-- Controllo che il profilo esista (solo per delete profilo, e
update del nome)
IF NOT p_is_insert THEN
    CALL sp_util_profilo_exists(p_nome_profilo, p_nome_progetto);
END IF;
END//


-- COMPONENTE: sp_componente_check USATO IN:
-- sp_componente_insert
-- sp_componente_delete
-- sp_componente_update

/*
* PROCEDURE: sp_componente_check
* PURPOSE: Controlla la validità di un componente.
*
* @param IN p_nome_componente - Nome del componente da controllare
* @param IN p_nome_progetto - Nome del progetto a cui appartiene il
componente
* @param IN p_email_creatore - Email dell'utente creatore del
progetto
* @param IN p_is_insert - Flag per distinguere tra insert, update e
delete
*
* @throws 45000 - COMPONENTE NON ESISTENTE (+ altri throw specifici
dalle sp_util utilizzate)
*/
CREATE PROCEDURE sp_componente_check(
    IN p_nome_componente VARCHAR(100),
    IN p_nome_progetto VARCHAR(100),
    IN p_email_creatore VARCHAR(100),
    IN p_is_insert BOOLEAN
)
BEGIN
    -- Controllo che il progetto esista
    CALL sp_util_progetto_exists(p_nome_progetto);

    -- Controllo che il progetto sia creato dall'utente
    CALL sp_util_creatore_is_progetto_owner(p_email_creatore,
p_nome_progetto);

    -- Controllo che il progetto sia di tipo hardware
    CALL sp_util_progetto_is_hardware(p_nome_progetto);

    -- Controllo che il progetto sia ancora aperto
    CALL sp_util_progetto_is_aperto(p_nome_progetto);

```

```

-- Controllo che il componente esista (solo per update e delete)
IF NOT p_is_insert THEN
    IF NOT EXISTS (SELECT 1
                    FROM COMPONENTE
                    WHERE nome_componente = p_nome_componente
                      AND nome_progetto = p_nome_progetto) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Componente non esistente.' ;
    END IF;
END IF;
END//


-- PARTECIPANTE: sp_partecipante_check USATO IN:
--   sp_partecipante_creatore_update (sp_partecipante_creatore_check)
--   sp_partecipante_utente_insert (sp_partecipante_utente_check)

/*
* PROCEDURE: sp_partecipante_check
* PURPOSE: Controlla la validità di un partecipante. I controlli sono
comuni a entrambe le stored procedure di seguito.
*
* @param IN p_nome_progetto - Nome del progetto a cui appartiene il
partecipante
* @param IN p_nome_profilo - Nome del profilo richiesto dal
partecipante
*/
CREATE PROCEDURE sp_partecipante_check(
    IN p_nome_progetto VARCHAR(100),
    IN p_nome_profilo VARCHAR(100)
)
BEGIN
    -- Controllo che il progetto esista
    CALL sp_util_progetto_exists(p_nome_progetto);

    -- Controllo che il progetto sia ancora aperto
    CALL sp_util_progetto_is_aperto(p_nome_progetto);

    -- Controllo che il progetto sia di tipo software
    CALL sp_util_progetto_is_software(p_nome_progetto);

    -- Controllo che il profilo esista
    CALL sp_util_profilo_exists(p_nome_profilo, p_nome_progetto);
END//


/*
* PROCEDURE: sp_partecipante_creatore_check
* PURPOSE: Controlla la validità di un partecipante. I controlli sono
specifici per il creatore del progetto.
*

```

```

/*
 * @param IN p_email_creatore - Email dell'utente creatore del
 * progetto
 * @param IN p_email_candidato - Email dell'utente candidato al
 * progetto
 * @param IN p_nome_progetto - Nome del progetto a cui il partecipante
 * si è candidato
 * @param IN p_nome_profilo - Nome del profilo richiesto dal
 * partecipante
 *
 * @throws 45000 - CANDIDATURA NON ESISTENTE (+ altri throw specifici
 * dalle sp_util utilizzate)
 */
CREATE PROCEDURE sp_partecipante_creatore_check(
    IN p_email_creatore VARCHAR(100),
    IN p_email_candidato VARCHAR(100),
    IN p_nome_progetto VARCHAR(100),
    IN p_nome_profilo VARCHAR(100)
)
BEGIN
    -- CONTROLLI: Vedi la documentazione di questo check
    CALL sp_partecipante_check(p_nome_progetto, p_nome_profilo);

    -- Controllo che l'utente sia il creatore del progetto
    CALL sp_util_creatore_is_progetto_owner(p_email_creatore,
    p_nome_progetto);

    -- Controllo che la candidatura esista
    IF NOT EXISTS (SELECT 1
                    FROM PARTECIPANTE
                    WHERE email_utente = p_email_candidato
                    AND nome_progetto = p_nome_progetto
                    AND nome_profilo = p_nome_profilo
                    AND stato = 'potenziale') THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Candidatura non esistente.';
    END IF;
END//


/*
 * PROCEDURE: sp_partecipante_utente_check
 * PURPOSE: Controlla la validità di un partecipante. I controlli sono
 * specifici per l'utente che si candida al progetto software.
 *
 * @param IN p_email - Email dell'utente che si candida al progetto
 * @param IN p_nome_progetto - Nome del progetto a cui il partecipante
 * si candida
 * @param IN p_nome_profilo - Nome del profilo richiesto dal
 * partecipante
 *
 * @throws 45000 - UTENTE CREATORE DEL PROGETTO

```

```

*  @throws 45000 - CANDIDATURA INSERITA PRECEDENTEMENTE
*          (+ altri throw specifici da sp_partecipante_check)
*/
CREATE PROCEDURE sp_partecipante_utente_check(
    IN p_email VARCHAR(100),
    IN p_nome_progetto VARCHAR(100),
    IN p_nome_profilo VARCHAR(100)
)
BEGIN
    -- Dichiarazione variabile per la competenza mancante/suo livello
    -- insufficiente (se esiste)
    DECLARE missing_skill VARCHAR(100) DEFAULT NULL;

    -- CONTROLLI: Vedi la documentazione di questo check
    CALL sp_partecipante_check(p_nome_progetto, p_nome_profilo);

    -- Controllo che l'utente NON sia il creatore del progetto
    IF EXISTS (SELECT 1
        FROM PROGETTO
        WHERE nome = p_nome_progetto
            AND email_creatore = p_email) THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Non puoi candidarti al tuo stesso
progetto.';
    END IF;

    -- Controllo che il profilo disponga di competenze richieste (che
    -- non sia un profilo vuoto/in formazione dal creatore)
    IF NOT EXISTS (SELECT 1
        FROM SKILL_PROFILO
        WHERE nome_profilo = p_nome_profilo
            AND nome_progetto = p_nome_progetto
            AND competenza IS NOT NULL
            AND competenza != '') THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Profilo senza competenze definite.
Attendi che il creatore aggiunga le competenze richieste.';
    END IF;

    -- Controllo che il profilo non sia già stato occupato da un altro
    -- utente
    IF EXISTS (SELECT 1
        FROM PARTECIPANTE
        WHERE nome_progetto = p_nome_progetto
            AND nome_profilo = p_nome_profilo
            AND stato = 'accettato') THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Profilo già\'' occupato da un altro
utente.';
    END IF;

```

```

-- Controllo se l'utente è stato precedentemente rifiutato per
questo profilo
IF EXISTS (SELECT 1
            FROM PARTECIPANTE
            WHERE email_utente = p_email
              AND nome_progetto = p_nome_progetto
              AND nome_profilo = p_nome_profilo
              AND stato = 'rifiutato') THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Utente precedentemente rifiutato per
questo profilo.';
END IF;

-- Controllo che l'utente non abbia già una candidatura per quel
profilo del progetto
IF EXISTS (SELECT 1
            FROM PARTECIPANTE
            WHERE email_utente = p_email
              AND nome_progetto = p_nome_progetto
              AND nome_profilo = p_nome_profilo) THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Candidatura già\` presente per questo
profilo. Attendi la risposta del creatore.';
END IF;

-- Per ogni competenza richiesta dal profilo, controlla che il
candidato abbia una entry in SKILL_CURRICULUM
-- con un livello_effettivo maggiore o uguale al
livello_richiesto.
SELECT sp.competenza
INTO missing_skill
FROM SKILL_PROFIL0 sp
      LEFT JOIN SKILL_CURRICULUM sc
              ON sp.competenza = sc.competenza AND
sc.email_utente = p_email
WHERE sp.nome_profilo = p_nome_profilo
  AND sp.nome_progetto = p_nome_progetto
  AND (sc.livello_effettivo IS NULL OR sc.livello_effettivo <
sp.livello_richiesto)
LIMIT 1;

-- Se almeno una competenza problematica viene trovata, lancia un
errore.
IF missing_skill IS NOT NULL THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Competenza mancante o livello
insufficiente. Assicurati di avere tutte le competenze richieste.';
END IF;
END//
```

```

-- COMMENTO: sp_commento_check e sp_commento_risposta_check USATI IN:
-- sp_commento_insert (sp_commento_check)
-- sp_commento_delete (sp_commento_check)
-- sp_commento_risposta_insert (sp_commento_risposta_check)
-- sp_commento_risposta_delete (sp_commento_risposta_check)

/*
* PROCEDURE: sp_commento_check
* PURPOSE: Controlla la validità di un commento.
*
* @param IN p_id - ID del commento da controllare
* @param IN p_nome_progetto - Nome del progetto a cui appartiene il
commento
* @param IN p_email_autore - Email dell'autore del commento
* @param IN p_is_insert - Flag per distinguere tra insert e delete
*
* @throws 45000 - NON SEI AUTORIZZATO A CANCELLARE QUESTO COMMENTO (+
altri throw specifici dalle sp_util utilizzate)
*/
CREATE PROCEDURE sp_commento_check(
    IN p_id INT,
    IN p_nome_progetto VARCHAR(100),
    IN p_email_autore VARCHAR(100),
    IN p_is_insert BOOLEAN
)
BEGIN
    -- Controllo che il progetto esista
    CALL sp_util_progetto_exists(p_nome_progetto);

    IF NOT p_is_insert THEN
        -- Controllo che il commento esista
        CALL sp_util_commento_exists(p_id);

        -- Controllo che l'utente sia l'autore del commento, OPPURE un
admin
        IF NOT (
            EXISTS (SELECT 1 FROM COMMENTO WHERE id = p_id AND
email_utente = p_email_autore)
            OR
            EXISTS (SELECT 1 FROM ADMIN WHERE email_utente =
p_email_autore)
        ) THEN
            SIGNAL SQLSTATE '45000'
                SET MESSAGE_TEXT = 'Non sei autorizzato a cancellare
questo commento.' ;
        END IF;
    END IF;
END//
```

```

/*
* PROCEDURE: sp_commento_risposta_check
* PURPOSE: Controlla la validità di una risposta a un commento.
*
* @param IN p_commento_id - ID del commento a cui si vuole
* rispondere/cancellare la risposta
* @param IN p_nome_progetto - Nome del progetto a cui appartiene il
commento
* @param IN p_email_creatore - Email dell'utente creatore del
progetto
* @param IN p_is_insert - Flag per distinguere tra insert e delete
*
* @throws 45000 - COMMENTO CONTIENE RISPOSTA
* @throws 45000 - UTENTE NON CREATORE O ADMIN (CANCELLAZIONE
RISPOSTA)
* @throws 45000 - COMMENTO NON CONTIENE RISPOSTA
* (+ altri throw specifici dalle sp_util utilizzate)
*/
CREATE PROCEDURE sp_commento_risposta_check(
    IN p_commento_id INT,
    IN p_nome_progetto VARCHAR(100),
    IN p_email_creatore VARCHAR(100),
    IN p_is_insert BOOLEAN
)
BEGIN
    -- Controllo che il progetto e il commento esistano
    CALL sp_util_progetto_exists(p_nome_progetto);
    CALL sp_util_commento_exists(p_commento_id);

    -- Se l'intento è di inserire una risposta...
    IF p_is_insert THEN
        -- Controllo che l'utente sia il creatore del progetto
        CALL sp_util_creatore_is_progetto_owner(p_email_creatore,
p_nome_progetto);
        -- Controllo che il commento NON ABBIA una risposta
        IF EXISTS (SELECT 1 FROM COMMENTO WHERE id = p_commento_id AND
risposta IS NOT NULL) THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Il commento contiene già una
risposta.';
        END IF;
    ELSE
        -- Altrimenti si intende cancellare una risposta...
        IF NOT ( -- L'utente deve essere il creatore del progetto o un
admin
            EXISTS (SELECT 1 FROM PROGETTO WHERE nome =
p_nome_progetto AND email_creatore = p_email_creatore)
            OR
            EXISTS (SELECT 1 FROM ADMIN WHERE email_utente =
p_email_creatore)
        )
    END IF;
END

```

```

        ) THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Non sei autorizzato a cancellare
questa risposta.';
        END IF;

        -- Controllo che il commento ABBIA una risposta da cancellare
        IF EXISTS (SELECT 1 FROM COMMENTO WHERE id = p_commento_id AND
risposta IS NULL) THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Il commento non contiene una
risposta.';
        END IF;
    END IF;
END//


-- FOTO: sp_foto_check USATO IN:
-- sp_foto_insert
-- sp_foto_delete

/*
* PROCEDURE: sp_foto_check
* PURPOSE: Controlla la validità di una foto.
*
* @param IN p_nome_progetto - Nome del progetto a cui appartiene la
foto
* @param IN p_email_creatore - Email dell'utente creatore del
progetto
* @param IN p_foto_id - ID della foto da controllare
* @param IN p_is_insert - Flag per distinguere tra insert e delete
*
* @throws 45000 - FOTO NON ESISTENTE (+ altri throw specifici da
sp_util_creatore_is_progetto_owner)
*/
CREATE PROCEDURE sp_foto_check(
    IN p_nome_progetto VARCHAR(100),
    IN p_email_creatore VARCHAR(100),
    IN p_foto_id INT,
    IN p_is_insert BOOLEAN
)
BEGIN
    -- Controllo che il progetto esista
    CALL sp_util_progetto_exists(p_nome_progetto);

    -- Controllo che l'utente sia il creatore del progetto
    CALL sp_util_creatore_is_progetto_owner(p_email_creatore,
p_nome_progetto);

    -- Controllo che il progetto sia aperto
    CALL sp_util_progetto_is_aperto(p_nome_progetto);

```

```

-- Controllo che la foto esista (solo per delete)
IF NOT p_is_insert AND NOT EXISTS (SELECT 1
                                    FROM FOTO
                                    WHERE nome_progetto =
p_nome_progetto
                                         AND id = p_foto_id) THEN
    SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Foto non esistente.';
END IF;
END//


DELIMITER ;

-- =====
-- STORED PROCEDURES (MAIN)
-- =====

-- In questo blocco vengono definite le stored procedure principali
-- ("main"). Implementano le funzionalità richieste dal progetto, con
-- qualche aggiunta.
-- Vengono divise in base alla tabella di riferimento, con la seguente
-- sintassi generale...

-- NOME_TABELLA:
-- sp_nome_tabella_{insert|delete|update|select} o altre azioni
-- specifiche

-- Al di sopra della definizione di ogni stored procedure principale,
-- si documenta la funzione di esse e che tipo di utente può utilizzarle.
-- L'interno di ogni stored procedure è diviso in due parti
-- principali:
-- 1. Controllo dei parametri in input (il controllo assistito dalle
-- stored procedure "helper")
-- 2. Operazioni sul database relative

DELIMITER //


-- UTENTE:
-- sp_utente_register
-- sp_utente_login
-- sp_utente_select

/*
* PROCEDURE: sp_utente_register
* PURPOSE: Registrazione di un utente con o senza ruolo di creatore,
* e/o admin.
* USED BY: ALL
*
* @param IN p_email - Email dell'utente

```

```

* @param IN p_password - Password dell'utente
* @param IN p_nickname - Nickname dell'utente
* @param IN p_nome - Nome dell'utente
* @param IN p_cognome - Cognome dell'utente
* @param IN p_anno_nascita - Anno di nascita dell'utente
* @param IN p_luogo_nascita - Luogo di nascita dell'utente
* @param IN p_is_creatore - Flag che indica se l'utente è un creatore
*/
CREATE PROCEDURE sp_utente_register(
    IN p_email VARCHAR(100),
    IN p_password VARCHAR(255),
    IN p_nickname VARCHAR(50),
    IN p_nome VARCHAR(50),
    IN p_cognome VARCHAR(50),
    IN p_anno_nascita INT,
    IN p_luogo_nascita VARCHAR(50),
    IN p_is_creatore BOOLEAN,
    IN p_is_admin BOOLEAN,
    IN p_codice_sicurezza VARCHAR(100)
)
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN
            ROLLBACK;
            RESIGNAL;
        END;
    START TRANSACTION;

    -- Controllo che l'utente non esista già
    IF EXISTS (SELECT 1 FROM UTENTE WHERE email = p_email) THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Email già registrata.';
    END IF;

    -- Controllo che l'utente sia maggiorenne
    IF p_anno_nascita > YEAR(CURRENT_DATE()) - 18 THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Devi essere maggiorenne per
registrarti.';
    END IF;

    INSERT INTO UTENTE (email, password, nickname, nome, cognome,
    anno_nascita, luogo_nascita)
        VALUES (p_email, p_password, p_nickname, p_nome, p_cognome,
    p_anno_nascita, p_luogo_nascita);

    IF p_is_creatore THEN
        INSERT INTO CREATORE (email_utente)
            VALUES (p_email);
    END IF;

```

```

IF p_is_admin THEN
    INSERT INTO ADMIN (email_utente, codice_sicurezza)
    VALUES (p_email, p_codice_sicurezza);
END IF;
COMMIT;
END//


/*
* PROCEDURE: sp_utente_login
* PURPOSE: Login di un utente, restituisce i dati dell'utente se
esiste.
* USED BY: ALL
*
* @param IN p_email - Email dell'utente
*/
CREATE PROCEDURE sp_utente_login(
    IN p_email VARCHAR(100)
)
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN
            ROLLBACK;
            RESIGNAL;
        END;
    START TRANSACTION;

    -- Controllo che l'utente esista
    IF NOT EXISTS (SELECT 1 FROM UTENTE WHERE email = p_email) THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Email non valida.';
    END IF;

    -- OK, restituisco i dati dell'utente
    SELECT nickname, email, password
    FROM UTENTE
    WHERE email = p_email;
    COMMIT;
END//


/*
* PROCEDURE: sp_utente_select
* PURPOSE: Visualizzazione dei dati di un utente (home.php).
* USED BY: ALL
*
* @param IN p_email - Email dell'utente
*/
CREATE PROCEDURE sp_utente_select(
    IN p_email VARCHAR(100)
)

```

```

BEGIN
    START TRANSACTION;
    SELECT nome, cognome, nickname, anno_nascita, luogo_nascita
    FROM UTENTE
    WHERE email = p_email;
    COMMIT;
END//


-- SKILL_CURRICULUM:
-- sp_skill_curriculum_insert
-- sp_skill_curriculum_update
-- sp_skill_curriculum_delete
-- sp_skill_curriculum_selectAll
-- sp_skill_curriculum_selectDiff

/*
* PROCEDURE: sp_skill_curriculum_insert
* PURPOSE: Inserimento di una skill in un curriculum utente.
* USED BY: ALL
*
* @param IN p_email - Email dell'utente
* @param IN p_competenza - Competenza da inserire nel curriculum
* @param IN p_livello - Livello della competenza da inserire (da 0 a
5)
*/
CREATE PROCEDURE sp_skill_curriculum_insert(
    IN p_email VARCHAR(100),
    IN p_competenza VARCHAR(100),
    IN p_livello TINYINT
)
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN
            ROLLBACK;
            RESIGNAL;
        END;
    START TRANSACTION;

    -- Controllo che la skill esista
    IF NOT EXISTS (SELECT 1 FROM SKILL WHERE competenza =
p_competenza) THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Skill non esistente.';
    END IF;

    -- Controllo che la skill non esista già nel curriculum
    dell'utente
    IF EXISTS (SELECT 1 FROM SKILL_CURRICULUM WHERE email_utente =
p_email AND competenza = p_competenza) THEN
        SIGNAL SQLSTATE '45000'

```

```

        SET MESSAGE_TEXT = 'Skill già\` presente nel curriculum.' ;
    END IF;

    -- Controllo che il livello sia valido
    IF p_livello < 0 OR p_livello > 5 THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Livello non valido. Inserisci un
valore tra 0 e 5.' ;
    END IF;

    -- OK, inserisco la skill nel curriculum
    INSERT INTO SKILL_CURRICULUM (email_utente, competenza,
livello_effettivo)
        VALUES (p_email, p_competenza, p_livello);
    COMMIT;
END//


/*
*  PROCEDURE: sp_skill_curriculum_update
*  PURPOSE: Aggiornamento del livello di una skill nel curriculum di
un utente.
*  USED BY: ALL
*
*  @param IN p_email - Email dell'utente
*  @param IN p_competenza - Competenza da aggiornare
*  @param IN p_livello - Nuovo livello della competenza (da 0 a 5)
*/
CREATE PROCEDURE sp_skill_curriculum_update(
    IN p_email VARCHAR(100),
    IN p_competenza VARCHAR(100),
    IN p_livello TINYINT
)
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN
            ROLLBACK;
            RESIGNAL;
        END;
    START TRANSACTION;

    -- Controllo che la skill esista
    IF NOT EXISTS (SELECT 1 FROM SKILL WHERE competenza =
p_competenza) THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Skill non esistente.' ;
    END IF;

    -- Controllo che la skill esista nel curriculum dell'utente
    IF NOT EXISTS (SELECT 1 FROM SKILL_CURRICULUM WHERE email_utente =
p_email AND competenza = p_competenza) THEN

```

```

        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Skill non presente nel curriculum.';
    END IF;

    -- Controllo che il livello sia valido
    IF p_livello < 0 OR p_livello > 5 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Livello non valido. Inserisci un
valore tra 0 e 5.';
    END IF;

    -- SIDE EFFECT: Rifiuto automaticamente le candidature che non
    soddisfano il livello o la competenza richiesta
    UPDATE PARTECIPANTE P
        JOIN SKILL_PROFIL0 SP ON P.nome_progetto =
SP.nome_progetto AND P.nome_profilo = SP.nome_profilo
        SET stato = 'rifiutato'
        WHERE P.email_utente = p_email
        AND SP.competenza = p_competenza
        AND SP.livello_richiesto > p_livello
        AND P.stato IN ('potenziale', 'accettato');

    -- OK, aggiorno il livello della skill
    UPDATE SKILL_CURRICULUM
    SET livello_effettivo = p_livello
    WHERE email_utente = p_email
        AND competenza = p_competenza;
    COMMIT;
END//


/*
* PROCEDURE: sp_skill_curriculum_delete
* PURPOSE: Rimozione di una skill dal curriculum di un utente.
* USED BY: ALL
* NOTE: Se l'utente è un partecipante potenziale/accettato a un
progetto che richiede questa skill,
*       la candidatura viene automaticamente rifiutata.
*
* @param IN p_email - Email dell'utente
* @param IN p_competenza - Competenza da rimuovere dal curriculum
*/
CREATE PROCEDURE sp_skill_curriculum_delete(
    IN p_email VARCHAR(100),
    IN p_competenza VARCHAR(100)
)
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        RESIGNAL;
    END;

```

```

        END;
        START TRANSACTION;

        -- Controllo che la skill esista
        IF NOT EXISTS (SELECT 1 FROM SKILL WHERE competenza =
p_competenza) THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Skill non esistente.';
        END IF;

        -- Controllo che la skill esista nel curriculum dell'utente
        IF NOT EXISTS (SELECT 1 FROM SKILL_CURRICULUM WHERE email_utente =
p_email AND competenza = p_competenza) THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Skill non presente nel curriculum.';
        END IF;

        -- SIDE EFFECT: Rifiuto automaticamente le candidature che non
soddisfano il livello o la competenza richiesta
        UPDATE PARTECIPANTE P
            JOIN SKILL_PROFILO SP ON P.nome_progetto =
SP.nome_progetto AND P.nome_profilo = SP.nome_profilo
            SET stato = 'rifiutato'
            WHERE P.email_utente = p_email
            AND SP.competenza = p_competenza
            AND P.stato IN ('potenziale', 'accettato');

        -- Rimuovo la skill dal curriculum dell'utente
        DELETE
        FROM SKILL_CURRICULUM
        WHERE email_utente = p_email
        AND competenza = p_competenza;
        COMMIT;
    END//


/*
*  PROCEDURE: sp_skill_curriculum_selectAll
*  PURPOSE: Visualizzazione di tutte le skill di un utente.
*  USED BY: ALL
*
*  @param IN p_email - Email dell'utente
*/
CREATE PROCEDURE sp_skill_curriculum_selectAll(
    IN p_email VARCHAR(100)
)
BEGIN
    START TRANSACTION;
    SELECT competenza, livello_effettivo
    FROM SKILL_CURRICULUM
    WHERE email_utente = p_email;

```

```

        COMMIT;
    END//


/*
*  PROCEDURE: sp_skill_curriculum_selectDiff
*  PURPOSE: Visualizzazione delle skill globali di cui un utente non
dispone/non ha inserito nel proprio curriculum.
*  USED BY: ALL
*
*  @param IN p_email - Email dell'utente
*/
CREATE PROCEDURE sp_skill_curriculum_selectDiff(
    IN p_email VARCHAR(100)
)
BEGIN
    START TRANSACTION;
    SELECT competenza
    FROM SKILL
    WHERE competenza NOT IN (SELECT competenza
                                FROM SKILL_CURRICULUM
                                WHERE email_utente = p_email);
    COMMIT;
END//


-- PROGETTO:
-- sp_progetto_select
-- sp_progetto_selectAll
-- sp_progetto_selectByCreatore
-- sp_progetto_insert
-- sp_progetto_descrizione_update
-- sp_progetto_budget_update


/*
*  PROCEDURE: sp_progetto_select
*  PURPOSE: Visualizzazione di un progetto specifico.
*  USED BY: ALL
*
*  @param IN p_nome - Nome del progetto
*/
CREATE PROCEDURE sp_progetto_select(
    IN p_nome VARCHAR(100)
)
BEGIN
    SELECT *
    FROM PROGETTO
    WHERE nome = p_nome;
END//


/*
*  PROCEDURE: sp_progetto_selectAll

```

```

* PURPOSE: Visualizzazione di tutti i progetti disponibili.
* USED BY: ALL
*/
CREATE PROCEDURE sp_progetto_selectAll()
BEGIN
    SELECT * FROM PROGETTO;
END//


/*
* PROCEDURE: sp_progetto_selectByCreatore
* PURPOSE: Visualizzazione di tutti i progetti di un creatore.
* USED BY: CREATORE
*
* @param IN p_email - Email del creatore
*/
CREATE PROCEDURE sp_progetto_selectByCreatore(
    IN p_email VARCHAR(100)
)
BEGIN
    SELECT *
    FROM PROGETTO
    WHERE email_creatore = p_email;
END//


/*
* PROCEDURE: sp_progetto_insert
* PURPOSE: Inserimento di un nuovo progetto da parte di un creatore.
* USED BY: CREATORE
* NOTE: Il campo CREATORE.nr_progetti_creati viene incrementato
        tramite trigger (trg_incremente_progetti_creati) dopo l'inserimento
        del progetto.
*
* @param IN p_nome - Nome del progetto
* @param IN p_email_creatore - Email del creatore del progetto
* @param IN p_descrizione - Descrizione del progetto
* @param IN p_budget - Budget del progetto
* @param IN p_data_limite - Data limite per il progetto
* @param IN p_tipo - Tipo di progetto (software o hardware)
*/
CREATE PROCEDURE sp_progetto_insert(
    IN p_nome VARCHAR(100),
    IN p_email_creatore VARCHAR(100),
    IN p_descrizione TEXT,
    IN p_budget DECIMAL(10, 2),
    IN p_data_limite DATE,
    IN p_tipo ENUM ('software', 'hardware')
)
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN

```

```

        ROLLBACK;
        RESIGNAL;
    END;
START TRANSACTION;
-- Controllo che l'utente sia un creatore
CALL sp_util_utente_is_creatore(p_email_creatore);

-- Controllo che il progetto non esista già
IF EXISTS (SELECT 1 FROM PROGETTO WHERE nome = p_nome) THEN
    SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Un progetto con questo nome esiste
gia\.';
END IF;

-- Controllo che la data_limite sia futura alla data attuale
IF p_data_limite <= CURRENT_DATE THEN
    SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Data limite deve essere futura alla
data attuale.';
END IF;

-- Controllo che il tipo di progetto sia valido
IF p_tipo NOT IN ('software', 'hardware') THEN
    SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Il tipo di progetto deve essere
software o hardware.';
END IF;

-- Controllo che il budget sia maggiore di 0
IF p_budget <= 0 THEN
    SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Il budget deve essere maggiore di 0.';
END IF;

INSERT INTO PROGETTO (nome, email_creatore, descrizione, budget,
data_limite)
VALUES (p_nome, p_email_creatore, p_descrizione, p_budget,
p_data_limite);

-- Insert in tabella specifica in base al tipo di progetto
IF p_tipo = 'software' THEN
    INSERT INTO PROGETTO_SOFTWARE (nome_progetto)
    VALUES (p_nome);
ELSEIF p_tipo = 'hardware' THEN
    INSERT INTO PROGETTO_HARDWARE (nome_progetto)
    VALUES (p_nome);
END IF;
COMMIT;
END//
```

```

/*
*  PROCEDURE: sp_progetto_descrizione_update
*  PURPOSE: Aggiornamento della descrizione di un progetto.
*  USED BY: CREATORE
*
*  @param IN p_nome - Nome del progetto
*  @param IN p_email_creatore - Email del creatore del progetto
*  @param IN p_descrizione - Nuova descrizione del progetto
*/
CREATE PROCEDURE sp_progetto_descrizione_update(
    IN p_nome VARCHAR(100),
    IN p_email_creatore VARCHAR(100),
    IN p_descrizione TEXT
)
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN
            ROLLBACK;
            RESIGNAL;
        END;
    START TRANSACTION;
    -- Controllo che il progetto esista
    CALL sp_util_progetto_exists(p_nome);

    -- Controllo che l'utente sia il creatore del progetto
    CALL sp_util_creatore_is_progetto_owner(p_email_creatore, p_nome);

    -- Controllo che il progetto sia aperto
    CALL sp_util_progetto_is_aperto(p_nome);

    -- Controllo che la descrizione non sia vuota
    IF p_descrizione IS NULL OR LENGTH(p_descrizione) < 1 THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'La descrizione non puo\' essere
vuota.';
    END IF;

    -- OK, aggiorno la descrizione
    UPDATE PROGETTO
    SET descrizione = p_descrizione
    WHERE nome = p_nome;
    COMMIT;
END//


/*
*  PROCEDURE: sp_progetto_budget_update
*  PURPOSE: Aggiornamento del budget di un progetto.
*  USED BY: CREATORE
*
*  @param IN p_nome - Nome del progetto

```

```

*  @param IN p_email_creatore - Email del creatore del progetto
*  @param IN p_budget - Nuovo budget del progetto
*/
CREATE PROCEDURE sp_progetto_budget_update(
    IN p_nome VARCHAR(100),
    IN p_email_creatore VARCHAR(100),
    IN p_budget DECIMAL(10, 2)
)
BEGIN
    DECLARE current_budget DECIMAL(10, 2);
    DECLARE current_state VARCHAR(20);
    DECLARE tot_finanziamento DECIMAL(10, 2);
    DECLARE tot_componenti DECIMAL(10, 2);

    DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN
            ROLLBACK;
            RESIGNAL;
        END;
    START TRANSACTION;
    -- Controllo che l'utente sia il creatore del progetto
    CALL sp_util_creatore_is_progetto_owner(p_email_creatore, p_nome);

    -- Recupero il budget corrente e lo stato del progetto
    SELECT budget, stato
    INTO current_budget, current_state
    FROM PROGETTO
    WHERE nome = p_nome;

    -- Controllo che il progetto non sia chiuso
    IF current_state != 'aperto' THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Impossibile modificare il budget di un
progetto chiuso.';
    END IF;

    -- Mi assicuro che il nuovo budget sia maggiore di 0
    IF p_budget <= 0 THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Il budget deve essere maggiore di 0.';
    END IF;

    -- Per i progetti hardware, verifico che il nuovo budget sia
    almeno uguale al costo totale dei componenti
    IF EXISTS(SELECT 1 FROM PROGETTO_HARDWARE WHERE nome_progetto =
p_nome) THEN
        SELECT IFNULL(SUM(prezzo * quantita), 0)
        INTO tot_componenti
        FROM COMPONENTE
        WHERE nome_progetto = p_nome;
    END IF;

```

```

    IF p_budget < tot_componenti THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Il budget deve essere almeno pari
al costo totale dei componenti.';
    END IF;
END IF;

-- Se il budget viene modificato, controllo il totale dei
finanziamenti
IF p_budget != current_budget THEN
    SELECT IFNULL(SUM(importo), 0)
    INTO tot_finanziamento
    FROM FINANZIAMENTO
    WHERE nome_progetto = p_nome;

-- Se la somma finanziamenti è >= al nuovo budget imposto lo
stato a 'chiuso'
IF tot_finanziamento >= p_budget THEN
    UPDATE PROGETTO
    SET stato = 'chiuso'
    WHERE nome = p_nome;
    COMMIT;
END IF;
END IF;

-- Se tutti i controlli passano, aggiorno il budget del progetto
UPDATE PROGETTO
SET budget = p_budget
WHERE nome = p_nome;
COMMIT;
END//


-- FINANZIAMENTO:
-- sp_finanziamento_insert
-- sp_finanziamento_selectSumByProgetto
-- sp_finanziamento_selectAllByProgetto
-- sp_finanziamento_selectAllByUtente

/*
* PROCEDURE: sp_finanziamento_insert
* PURPOSE: Finanziamento di un progetto da parte di un utente.
* USED BY: ALL
* NOTE:
*   - Il progetto deve essere aperto
*   - Il codice della reward è scelto dall'utente al livello di
interfaccia al momento del finanziamento, in base all'importo scelto
*   - Anche il creatore può finanziare il proprio progetto
*
* @param IN p_email - Email dell'utente

```

```

*  @param IN p_nome_progetto - Nome del progetto
*  @param IN p_codice_reward - Codice del reward scelto dall'utente
*  @param IN p_importo - Importo del finanziamento
*/
CREATE PROCEDURE sp_finanziamento_insert(
    IN p_email VARCHAR(100),
    IN p_nome_progetto VARCHAR(100),
    IN p_codice_reward VARCHAR(50), -- Scelto al livello di
interfaccia
    IN p_importo DECIMAL(10, 2)
)
BEGIN
    DECLARE num_finanziamenti INT;

    DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN
            ROLLBACK;
            RESIGNAL;
        END;
    START TRANSACTION;

    -- Controllo che il progetto sia aperto
    CALL sp_util_progetto_is_aperto(p_nome_progetto);

    -- Controllo che l'utente non abbia già effettuato un
finanziamento nello stesso giorno
    SELECT COUNT(*)
    INTO num_finanziamenti
    FROM FINANZIAMENTO
    WHERE email_utente = p_email
        AND nome_progetto = p_nome_progetto
        AND data = CURDATE();

    IF num_finanziamenti > 0 THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Finanziamento già\` effettuato oggi.' ;
    END IF;

    -- Controllo che la reward esista
    IF NOT EXISTS (SELECT 1
                    FROM REWARD
                    WHERE codice = p_codice_reward
                        AND nome_progetto = p_nome_progetto) THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Reward selezionata inesistente.' ;
    END IF;

    -- Controllo che l'importo sia >= 0.01
    IF p_importo < 0.01 THEN
        SIGNAL SQLSTATE '45000'

```

```

        SET MESSAGE_TEXT = 'Importo deve essere maggiore o uguale
a 0.01.';
        END IF;

        -- Controllo che l'importo sia maggiore o uguale al minimo
        richiesto dalla reward
        CALL sp_util_reward_valid_finanziamento(p_nome_progetto,
p_codice_reward, p_importo);

        -- OK, inserisco il finanziamento
        INSERT INTO FINANZIAMENTO (email_utente, nome_progetto,
codice_reward, importo)
        VALUES (p_email, p_nome_progetto, p_codice_reward, p_importo);
        COMMIT;
END//


/*
* PROCEDURE: sp_finanziamento_selectSumByProgetto
* PURPOSE: Restituisce la somma degli importi dei finanziamenti
ricevuti da un progetto.
* USED BY: ALL
* NOTE: Se il progetto non ha ricevuto finanziamenti, restituisce 0
piuttosto che NULL.
*
* @param IN p_nome_progetto - Nome del progetto da cui si vuole
ottenere il totale dei finanziamenti
*/
CREATE PROCEDURE sp_finanziamento_selectSumByProgetto(
    IN p_nome_progetto VARCHAR(100)
)
BEGIN
    START TRANSACTION;
    SELECT IFNULL(SUM(importo), 0) AS totale_finanziamenti
    FROM FINANZIAMENTO
    WHERE nome_progetto = p_nome_progetto;
    COMMIT;
END//


/*
* PROCEDURE: sp_finanziamento_selectAllByProgetto
* PURPOSE: Restituisce tutti i finanziamenti ricevuti dai progetti
creati da un utente specifico.
* USED BY: CREATORE
*
* @param IN p_email_creatore - Email del creatore di cui si vogliono
ottenere i finanziamenti ricevuti
*/
CREATE PROCEDURE sp_finanziamento_selectAllByProgetto(
    IN p_email_creatore VARCHAR(100)
)

```

```

BEGIN
    START TRANSACTION;
    SELECT F.data,
           F.email_utente,
           U.nickname      AS finanziatore_nickname,
           F.nome_progetto,
           F.codice_reward,
           F.importo,
           R.descrizione AS reward_descrizione,
           R.foto         AS reward_foto,
           P.budget       AS progetto_budget,
           P.stato        AS progetto_stato
    FROM FINANZIAMENTO F
        JOIN PROGETTO P ON F.nome_progetto = P.nome
        JOIN UTENTE U ON F.email_utente = U.email
        JOIN REWARD R ON F.codice_reward = R.codice AND
F.nome_progetto = R.nome_progetto
    WHERE P.email_creatore = p_email_creatore
    ORDER BY F.data DESC;
    COMMIT;
END//
```

```

/*
* PROCEDURE: sp_finanziamento_selectAllByUtente
* PURPOSE: Restituisce tutti i finanziamenti effettuati da un utente
specifico con dettagli completi su reward e progetto.
* USED BY: ALL
*
* @param IN p_email - Email dell'utente da cui si vuole ottenere la
lista dei finanziamenti
*/
CREATE PROCEDURE sp_finanziamento_selectAllByUtente(
    IN p_email VARCHAR(100)
)
BEGIN
    START TRANSACTION;
    SELECT F.data,
           F.email_utente,
           F.nome_progetto,
           F.codice_reward,
           F.importo,
           R.descrizione AS reward_descrizione,
           R.foto         AS reward_foto,
           R.min_importo AS reward_min_importo,
           P.email_creatore,
           P.stato        AS progetto_stato,
           P.budget       AS progetto_budget,
           P.data_limite AS progetto_data_limite
    FROM FINANZIAMENTO F
        JOIN PROGETTO P ON F.nome_progetto = P.nome
```

```

        JOIN REWARD R ON F.codice_reward = R.codice AND
F.nome_progetto = R.nome_progetto
      WHERE F.email_utente = p_email
      ORDER BY F.data DESC;
      COMMIT;
END//


-- COMMENTO:
-- sp_commento_insert
-- sp_commento_delete
-- sp_commento_selectAll
-- sp_commento_risposta_insert
-- sp_commento_risposta_delete

/*
* PROCEDURE: sp_commento_insert
* PURPOSE: Inserimento di un commento a un progetto esistente.
* USED BY: ALL
*
* @param IN p_email_autore - Email dell'autore del commento
* @param IN p_nome_progetto - Nome del progetto
* @param IN p_testo - Testo del commento
*/
CREATE PROCEDURE sp_commento_insert(
  IN p_email_autore VARCHAR(100),
  IN p_nome_progetto VARCHAR(100),
  IN p_testo TEXT
)
BEGIN
  DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
      ROLLBACK;
      RESIGNAL;
    END;
  START TRANSACTION;
  -- Controllo che esista il progetto
  CALL sp_commento_check(NULL, p_nome_progetto, p_email_autore,
TRUE);

  -- Controllo che il commento sia almeno lungo 1 carattere
  IF p_testo IS NULL OR LENGTH(p_testo) < 1 THEN
    SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'Il commento non puo\' essere vuoto.';
  END IF;

  -- OK, inserisco il commento
  INSERT INTO COMMENTO (email_utente, nome_progetto, testo)
VALUES (p_email_autore, p_nome_progetto, p_testo);
  COMMIT;
END//

```

```

/*
* PROCEDURE: sp_commento_delete
* PURPOSE: Cancellazione di un commento a un progetto esistente.
* USED BY: ALL
* NOTE: L'utente può cancellare solo i propri commenti, mentre un
admin può cancellare qualsiasi commento.
*
* @param IN p_id - ID del commento da cancellare
* @param IN p_email - Email dell'utente (autore del commento o admin)
* @param IN p_nome_progetto - Nome del progetto del quale fa parte il
commento
*/
CREATE PROCEDURE sp_commento_delete(
    IN p_id INT,
    IN p_email VARCHAR(100),
    IN p_nome_progetto VARCHAR(100)
)
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN
            ROLLBACK;
            RESIGNAL;
        END;
    START TRANSACTION;
    -- CONTROLLI: Vedi la documentazione di questo check
    CALL sp_commento_check(p_id, p_nome_progetto, p_email, FALSE);

    -- OK, cancello il commento
    DELETE
        FROM COMMENTO
        WHERE id = p_id;
    COMMIT;
END//


/*
* PROCEDURE: sp_commento_selectAll
* PURPOSE: Visualizzazione di tutti i commenti di un progetto.
* USED BY: ALL
*
* @param IN p_nome_progetto - Nome del progetto
*/
CREATE PROCEDURE sp_commento_selectAll(
    IN p_nome_progetto VARCHAR(100)
)
BEGIN
    SELECT C.id, C.email_utente, U.nickname, C.testo, C.risposta,
C.data
        FROM COMMENTO C
        JOIN UTENTE U ON U.email = C.email_utente

```

```

        WHERE nome_progetto = p_nome_progetto;
END//


/*
* PROCEDURE: sp_commento_risposta_insert
* PURPOSE: Inserimento di una risposta a un commento esistente.
* USED BY: CREATORE
*
* @param IN p_commento_id - ID del commento a cui rispondere
* @param IN p_email_creatore - Email del creatore del progetto
* @param IN p_nome_progetto - Nome del progetto del quale fa parte il commento
* @param IN p_risposta - Testo della risposta
*/
CREATE PROCEDURE sp_commento_risposta_insert(
    IN p_commento_id INT,
    IN p_email_creatore VARCHAR(100),
    IN p_nome_progetto VARCHAR(100),
    IN p_risposta TEXT
)
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN
            ROLLBACK;
            RESIGNAL;
        END;
    START TRANSACTION;
    -- CONTROLLI: Vedi la documentazione di questo check
    CALL sp_commento_risposta_check(p_commento_id, p_nome_progetto,
p_email_creatore, TRUE);

    -- OK, inserisco la risposta
    UPDATE COMMENTO
    SET risposta = p_risposta
    WHERE id = p_commento_id;
    COMMIT;
END//


/*
* PROCEDURE: sp_commento_risposta_delete
* PURPOSE: Cancellazione di una risposta a un commento esistente.
* USED BY: CREATORE
* NOTE: Sia il creatore che un admin possono cancellare le risposte ai commenti del progetto suo (creatore).
*
* @param IN p_commento_id - ID del commento con risposta da cancellare
* @param IN p_email_creatore - Email del creatore del progetto
* @param IN p_nome_progetto - Nome del progetto del quale fa parte il commento

```

```

*/
CREATE PROCEDURE sp_commento_risposta_delete(
    IN p_commento_id INT,
    IN p_email_creatore VARCHAR(100),
    IN p_nome_progetto VARCHAR(100)
)
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN
            ROLLBACK;
            RESIGNAL;
        END;
    START TRANSACTION;
    -- CONTROLLI: Vedi la documentazione di questo check
    CALL sp_commento_risposta_check(p_commento_id, p_nome_progetto,
p_email_creatore, FALSE);

    -- OK, cancello la risposta
    UPDATE COMMENTO
    SET risposta = NULL
    WHERE id = p_commento_id;
    COMMIT;
END//


-- PARTECIPANTE:
-- sp_partecipante_utente_insert
-- sp_partecipante_creatore_update
-- sp_partecipante_selectAllAcceptedByProgetto
-- sp_partecipante_selectAllByUtente
-- sp_partecipante_selectAllByCreatore
-- sp_partecipante_getStatus

/*
* PROCEDURE: sp_partecipante_utente_insert
* PURPOSE: Inserimento di una candidatura a un progetto software da
parte di un utente.
* USED BY: ALL
* NOTE:
*   - L'utente non può candidarsi a un progetto di cui è creatore, e
non può candidarsi più di una volta per la stessa competenza.
*   - Utenti che non dispongono della competenza richiesta non possono
candidarsi.
*   - Utenti che non dispongono del livello richiesto per la
competenza non possono candidarsi e vengono automaticamente rifiutati
*       senza mai essere inseriti nella tabella PARTECIPANTE.
*
* @param IN p_email - Email dell'utente che si candida
* @param IN p_nome_progetto - Nome del progetto interessato
* @param IN p_nome_profilo - Nome del profilo richiesto per il
progetto

```

```

/*
CREATE PROCEDURE sp_partecipante_utente_insert(
    IN p_email VARCHAR(100),
    IN p_nome_progetto VARCHAR(100),
    IN p_nome_profilo VARCHAR(100)
)
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN
            ROLLBACK;
            RESIGNAL;
        END;
    START TRANSACTION;
    -- CONTROLLI: Vedi la documentazione di questo check
    CALL sp_partecipante_utente_check(p_email, p_nome_progetto,
p_nome_profilo);

    -- OK, inserisco la candidatura
    INSERT INTO PARTECIPANTE (email_utente, nome_progetto,
nome_profilo)
        VALUES (p_email, p_nome_progetto, p_nome_profilo);
    COMMIT;
END//


/*
*  PROCEDURE: sp_partecipante_creatore_update
*  PURPOSE: Accettazione o rifiuto di una candidatura a un progetto
software da parte del creatore.
*  USED BY: CREATORE
*  NOTE: Solo il creatore, in quanto tale, può accettare o rifiutare
una candidatura.
*
*  @param IN p_email_creatore - Email del creatore del progetto
*  @param IN p_email_candidato - Email del candidato
*  @param IN p_nome_progetto - Nome del progetto interessato
*  @param IN p_nome_profilo - Nome del profilo richiesto per il
progetto
*  @param IN p_nuovo_stato - Nuovo stato della candidatura (accettato
o rifiutato)
*/
CREATE PROCEDURE sp_partecipante_creatore_update(
    IN p_email_creatore VARCHAR(100),
    IN p_email_candidato VARCHAR(100),
    IN p_nome_progetto VARCHAR(100),
    IN p_nome_profilo VARCHAR(100),
    IN p_nuovo_stato ENUM ('accettato','rifiutato')
)
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN

```

```

        ROLLBACK;
        RESIGNAL;
    END;
    START TRANSACTION;
    -- CONTROLLI: Vedi la documentazione di questo check
    CALL sp_partecipante_creatore_check(p_email_creatore,
p_email_candidato, p_nome_progetto, p_nome_profilo);

    -- OK, aggiorno lo stato della candidatura
    UPDATE PARTECIPANTE
    SET stato = p_nuovo_stato
    WHERE email_utente = p_email_candidato
        AND nome_progetto = p_nome_progetto
        AND nome_profilo = p_nome_profilo;
    COMMIT;
END//


/*
*  PROCEDURE: sp_partecipante_selectAllAcceptedByProgetto
*  PURPOSE: Visualizzazione di tutti i partecipanti accettati per un
progetto.
*  USED BY: ALL
*
*  @param IN p_nome_progetto - Nome del progetto
*/
CREATE PROCEDURE sp_partecipante_selectAllAcceptedByProgetto(
    IN p_nome_progetto VARCHAR(100)
)
BEGIN
    START TRANSACTION;
    SELECT P.nome_profilo,
           P.email_utente,
           U.nickname
    FROM PARTECIPANTE P
        JOIN UTENTE U ON P.email_utente = U.email
    WHERE P.nome_progetto = p_nome_progetto
        AND P.stato = 'accettato';
    COMMIT;
END//


/*
*  PROCEDURE: sp_partecipante_selectAllByUtente
*  PURPOSE: Visualizzazione di tutte le candidature inviate da un
utente.
*  USED BY: ALL
*
*  @param IN p_email - Email dell'utente
*/
CREATE PROCEDURE sp_partecipante_selectAllByUtente(
    IN p_email VARCHAR(100)

```

```

)
BEGIN
    START TRANSACTION;
    SELECT P.email_utente,
           P.nome_progetto,
           P.nome_profilo,
           P.stato,
           PR.descrizione,
           PR.email_creatore,
           U.nickname AS creatoreNickname
    FROM PARTECIPANTE P
        JOIN PROGETTO PR ON P.nome_progetto = PR.nome
        JOIN UTENTE U ON PR.email_creatore = U.email
    WHERE P.email_utente = p_email
    ORDER BY P.stato, P.nome_progetto;
    COMMIT;
END//


/*
* PROCEDURE: sp_partecipante_selectAllByCreatore
* PURPOSE: Visualizzazione di tutte le candidature ricevute per i
progetti di un creatore.
* USED BY: CREATORE
*
* @param IN p_email - Email del creatore
*/
CREATE PROCEDURE sp_partecipante_selectAllByCreatore(
    IN p_email VARCHAR(100)
)
BEGIN
    START TRANSACTION;
    SELECT P.email_utente,
           P.nome_progetto,
           P.nome_profilo,
           P.stato,
           U.nickname AS candidatoNickname
    FROM PARTECIPANTE P
        JOIN PROGETTO PR ON P.nome_progetto = PR.nome
        JOIN UTENTE U ON P.email_utente = U.email
    WHERE PR.email_creatore = p_email
    ORDER BY P.stato, P.nome_progetto, P.nome_profilo;
    COMMIT;
END//


/*
* PROCEDURE: sp_partecipante_getStatus
* PURPOSE: Restituisce lo stato di partecipazione di un utente a un
profilo di un progetto.
* USED BY: ALL
*

```

```

*  @param IN p_email_utente - Email dell'utente
*  @param IN p_nome_progetto - Nome del progetto
*  @param IN p_nome_profilo - Nome del profilo
*/
CREATE PROCEDURE sp_partecipante_getStatus(
    IN p_email_utente VARCHAR(100),
    IN p_nome_progetto VARCHAR(100),
    IN p_nome_profilo VARCHAR(100)
)
BEGIN
    START TRANSACTION;
    SELECT stato
    FROM PARTECIPANTE
    WHERE email_utente = p_email_utente
        AND nome_progetto = p_nome_progetto
        AND nome_profilo = p_nome_profilo;
    COMMIT;
END//


-- SKILL:
-- sp_skill_insert
-- sp_skill_update
-- sp_skill_selectAll


/*
*  PROCEDURE: sp_skill_insert
*  PURPOSE: Inserimento di una competenza nella lista delle competenze
disponibili.
*  USED BY: ADMIN
*
*  @param IN p_competenza - Competenza/skill da inserire
*  @param IN p_email - Email dell'utente admin che esegue la procedura
*/
CREATE PROCEDURE sp_skill_insert(
    IN p_competenza VARCHAR(100),
    IN p_email VARCHAR(100)
)
BEGIN
DECLARE EXIT HANDLER FOR SQLEXCEPTION
BEGIN
    ROLLBACK;
    RESIGNAL;
END;
START TRANSACTION;
-- Controllo che l'admin sia l'utente che esegue la procedura
CALL sp_util_utente_is_admin(p_email);

-- Controllo che la competenza non esista già
IF EXISTS (SELECT 1 FROM SKILL WHERE competenza = p_competenza)
THEN

```

```

        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Skill già\ esistente';
END IF;

-- Controllo che la competenza non sia nulla o vuota
IF p_competenza IS NULL OR p_competenza = '' THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Skill non puo\ essere nulla o vuota';
END IF;

-- OK, inserisco la competenza
INSERT INTO SKILL (competenza)
VALUES (p_competenza);
COMMIT;

END//


/*
* PROCEDURE: sp_skill_update
* PURPOSE: Aggiornamento del nome di una skill globale.
* USED BY: ADMIN
*
* @param IN p_email_admin - Email dell'amministratore che esegue
l'aggiornamento
* @param IN p_vecchia_competenza - Nome attuale della competenza
* @param IN p_nuova_competenza - Nuovo nome della competenza
*/
CREATE PROCEDURE sp_skill_update(
    IN p_email_admin VARCHAR(100),
    IN p_vecchia_competenza VARCHAR(100),
    IN p_nuova_competenza VARCHAR(100)
)
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        RESIGNAL;
    END;
    START TRANSACTION;

    -- Controllo che l'utente sia un admin
    CALL sp_util_utente_is_admin(p_email_admin);

    -- Controllo che la vecchia skill esista
    IF NOT EXISTS (SELECT 1 FROM SKILL WHERE competenza =
p_vecchia_competenza) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Skill non esistente';
    END IF;

    -- Controllo che la nuova skill non esista già

```

```

        IF EXISTS (SELECT 1 FROM SKILL WHERE competenza =
p_nuova_competenza) THEN
            SIGNAL SQLSTATE '45000'
                SET MESSAGE_TEXT = 'Esiste già una skill con questo
nome';
        END IF;

        -- Controllo che la nuova skill non sia nulla o vuota
        IF p_nuova_competenza IS NULL OR TRIM(p_nuova_competenza) = ''
THEN
            SIGNAL SQLSTATE '45000'
                SET MESSAGE_TEXT = 'Skill non può essere nulla o vuota';
        END IF;

        -- OK, aggiorno il nome della skill
        UPDATE SKILL
        SET competenza = p_nuova_competenza
        WHERE competenza = p_vecchia_competenza;
        COMMIT;
    END//


/*
*  PROCEDURE: sp_skill_selectAll
*  PURPOSE: Visualizzazione di tutte le competenze disponibili nel
sistema.
*  USED BY: ALL
*/
CREATE PROCEDURE sp_skill_selectAll()
BEGIN
    START TRANSACTION;
    SELECT competenza
    FROM SKILL;
    COMMIT;
END//


-- REWARD:
-- sp_reward_insert
-- sp_reward_selectAllByProgetto
-- sp_reward_selectAllByFinanziamentoImporto

/*
*  PROCEDURE: sp_reward_insert
*  PURPOSE: Inserimento di una reward per un progetto.
*  USED BY: CREATORE
*  NOTE:
*      - p_codice è un codice univoco per la reward, definito dal
creatore al momento dell'inserimento
*      - p_min_importo è l'importo minimo per il quale un finanziatore è
eleggibile alla reward quando effettua un finanziamento
*

```

```

* @param IN p_codice - Codice della reward definito dal creatore
* @param IN p_nome_progetto - Nome del progetto a cui appartiene la
reward
* @param IN p_email_creatore - Email del creatore del progetto che
inserisce la reward
* @param IN p_descrizione - Descrizione della reward
* @param IN p_foto - Foto della reward
* @param IN p_min_importo - Importo minimo per essere eleggibili alla
reward
*/
CREATE PROCEDURE sp_reward_insert(
    IN p_codice VARCHAR(50),
    IN p_nome_progetto VARCHAR(100),
    IN p_email_creatore VARCHAR(100),
    IN p_descrizione TEXT,
    IN p_foto MEDIUMBLOB,
    IN p_min_importo DECIMAL(10, 2)
)
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN
            ROLLBACK;
            RESIGNAL;
        END;
    START TRANSACTION;
    -- Controllo che il progetto esista
    CALL sp_util_progetto_exists(p_nome_progetto);

    -- Controllo che l'utente sia il creatore del progetto
    CALL sp_util_creatore_is_progetto_owner(p_email_creatore,
p_nome_progetto);

    -- Controllo che il progetto sia aperto
    CALL sp_util_progetto_is_aperto(p_nome_progetto);

    -- Controllo che il codice della reward non sia già stato
utilizzato
    IF EXISTS(SELECT 1 FROM REWARD WHERE codice = p_codice AND
nome_progetto = p_nome_progetto) THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Codice reward gia\' utilizzato per
questo progetto.';
    END IF;

    -- Controllo che la descrizione non sia vuota
    IF p_descrizione IS NULL OR LENGTH(p_descrizione) < 1 THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'La descrizione non puo\' essere
vuota.';
    END IF;

```

```

-- Controllo che l'importo minimo sia >= 0.01
IF p_min_importo < 0.01 THEN
    SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Importo minimo deve essere maggiore o
uguale a 0.01.';
END IF;

-- Controllo che la foto sia valida
IF p_foto IS NULL THEN
    SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'La foto non puo\' essere vuota.';
END IF;

-- OK, inserisco la reward
INSERT INTO REWARD (codice, nome_progetto, descrizione, foto,
min_importo)
VALUES (p_codice, p_nome_progetto, p_descrizione, p_foto,
p_min_importo);
COMMIT;
END//


/*
* PROCEDURE: sp_reward_selectAllByProgetto
* PURPOSE: Visualizzazione di tutte le reward di un progetto.
* USED BY: ALL
*
* @param IN p_nome_progetto - Nome del progetto
*/
CREATE PROCEDURE sp_reward_selectAllByProgetto(
    IN p_nome_progetto VARCHAR(100)
)
BEGIN
START TRANSACTION;
-- Controllo che il progetto esista
CALL sp_util_progetto_exists(p_nome_progetto);

-- OK, restituisco le reward
SELECT codice, descrizione, foto, min_importo
FROM REWARD
WHERE nome_progetto = p_nome_progetto;
COMMIT;
END//


/*
* PROCEDURE: sp_reward_selectAllByFinanziamentoImporto
* PURPOSE: Visualizzazione di tutte le reward disponibili per un
progetto in base all'importo donato.
* Restituisce le reward per le quali l'importo donato è
maggiore o uguale al valore minimo richiesto (min_importo).

```

```

* USED BY: ALL
*
* @param IN p_nome_progetto - Nome del progetto
* @param IN p_importo - Importo del finanziamento effettuato
dall'utente
*/
CREATE PROCEDURE sp_reward_selectAllByFinanziamentoImporto(
    IN p_nome_progetto VARCHAR(100),
    IN p_importo DECIMAL(10, 2)
)
BEGIN
    START TRANSACTION;
    SELECT codice, descrizione, foto, min_importo
    FROM REWARD
    WHERE nome_progetto = p_nome_progetto
        AND min_importo <= p_importo;
    COMMIT;
END//


-- COMPONENTE:
-- sp_componente_insert
-- sp_componente_delete
-- sp_componente_update
-- sp_componente_selectAllByProgetto

/*
* PROCEDURE: sp_componente_insert
* PURPOSE: Inserimento di un componente per un progetto hardware.
* USED BY: CREATORE
* NOTE:
*      - Il prezzo e la quantità del componente vengono definiti dal
creatore al momento dell'inserimento.
*      - Il prezzo e la quantità del componente aggiorneranno il budget
del progetto se (prezzo * quantità) > budget attuale del progetto.
*
* @param IN p_nome_componente - Nome del componente da inserire per
il progetto hardware
* @param IN p_nome_progetto - Nome del progetto hardware
* @param IN p_descrizione - Descrizione del componente
* @param IN p_quantita - Quantità del componente
* @param IN p_prezzo - Prezzo del componente
* @param IN p_email_creatore - Email del creatore del progetto che
richiede l'inserimento
*/
CREATE PROCEDURE sp_componente_insert(
    IN p_nome_componente VARCHAR(100),
    IN p_nome_progetto VARCHAR(100),
    IN p_descrizione TEXT,
    IN p_quantita INT,
    IN p_prezzo DECIMAL(10, 2),

```

```

    IN p_email_creatore VARCHAR(100)
)
BEGIN
    DECLARE tot_componenti DECIMAL(10, 2);
    DECLARE budget_progetto DECIMAL(10, 2);
    DECLARE eccesso DECIMAL(10, 2);

    DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN
            ROLLBACK;
            RESIGNAL;
        END;
    START TRANSACTION;

    -- CONTROLLI: Vedi la documentazione di questo check
    CALL sp_componente_check(p_nome_componente, p_nome_progetto,
p_email_creatore, TRUE);

    -- Controllo che la quantità sia > 0
    IF p_quantita < 1 THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'La quantita\' deve essere maggiore di
0.';
    END IF;

    -- Recupero il budget del progetto
    SELECT budget
    INTO budget_progetto
    FROM PROGETTO
    WHERE nome = p_nome_progetto;

    -- Calcolo il costo totale dei componenti del progetto
    SELECT IFNULL(SUM(prezzo * quantita), 0)
    INTO tot_componenti
    FROM COMPONENTE
    WHERE nome_progetto = p_nome_progetto;

    -- Determino l'eccesso di budget
    SET eccesso = (tot_componenti + (p_prezzo * p_quantita)) -
budget_progetto;

    -- Se l'eccesso è > 0, aggiorno il budget del progetto
    IF eccesso > 0 THEN
        UPDATE PROGETTO
        SET budget = budget + eccesso
        WHERE nome = p_nome_progetto;
    END IF;

    -- OK, inserisco il componente
    INSERT INTO COMPONENTE (nome_componente, nome_progetto,

```

```

descrizione, quantita, prezzo)
    VALUES (p_nome_componente, p_nome_progetto, p_descrizione,
p_quantita, p_prezzo);
    COMMIT;
END//


/*
*  PROCEDURE: sp_componente_delete
*  PURPOSE: Rimozione di un componente per un progetto hardware.
*  USED BY: CREATORE
*  NOTE: Il budget del progetto viene aggiornato in base alla quantità
e al prezzo del componente rimosso.
*
*  @param IN p_nome_componente - Nome del componente da rimuovere
*  @param IN p_nome_progetto - Nome del progetto hardware a cui
appartiene il componente
*  @param IN p_email_creatore - Email del creatore del progetto che
richiede la rimozione
*/
CREATE PROCEDURE sp_componente_delete(
    IN p_nome_componente VARCHAR(100),
    IN p_nome_progetto VARCHAR(100),
    IN p_email_creatore VARCHAR(100)
)
BEGIN
    DECLARE budget_progetto DECIMAL(10, 2);
    DECLARE comp_prezzo DECIMAL(10, 2);
    DECLARE comp_quantita INT;
    DECLARE new_budget DECIMAL(10, 2);

    DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN
            ROLLBACK;
            RESIGNAL;
        END;
    START TRANSACTION;

    -- CONTROLLI: Vedi la documentazione di questo check
    CALL sp_componente_check(p_nome_componente, p_nome_progetto,
p_email_creatore, FALSE);

    -- Controllo che ci sia almeno un componente rimanente
    IF (SELECT COUNT(*) FROM COMPONENTE WHERE nome_progetto =
p_nome_progetto) = 1 THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Deve esserci almeno un componente per
il progetto';
        END IF;

    -- Recupera il budget del progetto

```

```

SELECT budget
INTO budget_progetto
FROM PROGETTO
WHERE nome = p_nome_progetto;

-- Recupera prezzo e quantità del componente da eliminare
SELECT prezzo, quantita
INTO comp_prezzo, comp_quantita
FROM COMPONENTE
WHERE nome_componente = p_nome_componente
AND nome_progetto = p_nome_progetto;

-- Determina il budget del progetto senza il componente rimosso
SET new_budget = budget_progetto - (comp_prezzo * comp_quantita);

-- Aggiorno il budget del progetto
UPDATE PROGETTO
SET budget = new_budget
WHERE nome = p_nome_progetto;

-- OK, rimuovo il componente
DELETE
FROM COMPONENTE
WHERE nome_componente = p_nome_componente
AND nome_progetto = p_nome_progetto;
COMMIT;

END//


/*
* PROCEDURE: sp_componente_update
* PURPOSE: Aggiornamento di un componente per un progetto hardware.
* USED BY: CREATORE
* NOTE:
*   - Il budget del progetto viene aggiornato in base alla differenza
*     tra il prezzo e la quantità del componente prima e dopo
*     l'aggiornamento.
*   - Se campi come nome e/o descrizione del componente non sono
*     modificati a livello di interfaccia, non vengono aggiornati e vengono
*     passati
*     nome e/o descrizione attuali.
*
*   @param IN p_nome_componente - Nome del componente da aggiornare
*   @param IN p_nuovo_nome_componente - Nuovo nome del componente (se
*     diverso da quello attuale)
*   @param IN p_nome_progetto - Nome del progetto hardware a cui
*     appartiene il componente
*   @param IN p_descrizione - Nuova descrizione del componente
*   @param IN p_quantita - Nuova quantità del componente
*   @param IN p_prezzo - Nuovo prezzo del componente
*   @param IN p_email_creatore - Email del creatore del progetto che

```

```

richiede l'aggiornamento
*/
CREATE PROCEDURE sp_componente_update(
    IN p_nome_componente VARCHAR(100),
    IN p_nuovo_nome_componente VARCHAR(100),
    IN p_nome_progetto VARCHAR(100),
    IN p_descrizione TEXT,
    IN p_quantita INT,
    IN p_prezzo DECIMAL(10, 2),
    IN p_email_creatore VARCHAR(100)
)
BEGIN
    DECLARE tot_componenti DECIMAL(10, 2);
    DECLARE budget_progetto DECIMAL(10, 2);
    DECLARE eccesso DECIMAL(10, 2);
    DECLARE old_totale DECIMAL(10, 2);
    DECLARE old_prezzo DECIMAL(10, 2);
    DECLARE old_quantita INT;

    DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN
            ROLLBACK;
            RESIGNAL;
        END;
    START TRANSACTION;

    -- CONTROLLI: Vedi la documentazione di questo check
    CALL sp_componente_check(p_nome_componente, p_nome_progetto,
    p_email_creatore, FALSE);

    -- Controllo che la quantità del componente sia > 0
    IF p_quantita <= 0 THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'La quantità del componente deve
essere maggiore di 0';
    END IF;

    -- Controllo che non esista già un componente con il nuovo nome
    IF p_nome_componente != p_nuovo_nome_componente AND
        EXISTS (SELECT 1
            FROM COMPONENTE
            WHERE nome_componente = p_nuovo_nome_componente
                AND nome_progetto = p_nome_progetto) THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Esiste già un componente con questo
nome';
    END IF;

    -- Recupero il budget del progetto
    SELECT budget

```

```

INTO budget_progetto
FROM PROGETTO
WHERE nome = p_nome_progetto;

-- Recupero il vecchio prezzo e quantità del componente
SELECT prezzo, quantita
INTO old_prezzo, old_quantita
FROM COMPONENTE
WHERE nome_componente = p_nome_componente
    AND nome_progetto = p_nome_progetto;

-- Calcolo il costo totale del componente prima dell'aggiornamento
SET old_totale = old_prezzo * old_quantita;

-- Calcolo il costo totale dei componenti del progetto, escludendo
il componente corrente
SELECT IFNULL(SUM(prezzo * quantita), 0)
INTO tot_componenti
FROM COMPONENTE
WHERE nome_progetto = p_nome_progetto
    AND NOT (nome_componente = p_nome_componente AND nome_progetto =
p_nome_progetto);

-- Determino l'eccesso di budget con il nuovo componente
SET eccesso = (tot_componenti + (p_prezzo * p_quantita)) -
budget_progetto;

-- Vale anche se l'"eccesso" è negativo, per ridurre il budget
UPDATE PROGETTO
SET budget = budget + eccesso
WHERE nome = p_nome_progetto;

-- OK, aggiorno il componente
UPDATE COMPONENTE
SET nome_componente = p_nuovo_nome_componente,
    descrizione = p_descrizione,
    quantita = p_quantita,
    prezzo = p_prezzo
WHERE nome_componente = p_nome_componente
    AND nome_progetto = p_nome_progetto;
COMMIT;

END//


/*
* PROCEDURE: sp_componente_selectAllByProgetto
* PURPOSE: Restituisce la lista dei componenti di un progetto
hardware.
* USED BY: ALL
*
* @param IN p_nome - Nome del progetto hardware di cui si vogliono

```

```

ottenere i componenti
*/
CREATE PROCEDURE sp_componente_selectAllByProgetto(
    IN p_nome VARCHAR(100)
)
BEGIN
    START TRANSACTION;
    SELECT nome_componente, descrizione, quantita, prezzo
    FROM COMPONENTE
    WHERE nome_progetto = p_nome;
    COMMIT;
END//


-- PROFILO:
-- sp_profilo_insert
-- sp_profilo_delete
-- sp_profilo_selectAllByProgetto
-- sp_profilo_nome_update

/*
*  PROCEDURE: sp_profilo_insert
*  PURPOSE: Inserimento di un profilo per un progetto software.
*  USED BY: CREATORE
*
*  @param IN p_nome_profilo - Nome del profilo da inserire
*  @param IN p_nome_progetto - Nome del progetto software a cui
appartiene il profilo
*  @param IN p_email_creatore - Email del creatore del progetto che
richiede l'inserimento
*/
CREATE PROCEDURE sp_profilo_insert(
    IN p_nome_profilo VARCHAR(100),
    IN p_nome_progetto VARCHAR(100),
    IN p_email_creatore VARCHAR(100)
)
BEGIN
DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        RESIGNAL;
    END;
    START TRANSACTION;
    -- CONTROLLI: Vedi la documentazione di questo check
    CALL sp_profilo_check(p_nome_profilo, p_email_creatore,
p_nome_progetto, TRUE);

    -- OK, inserisco il profilo
    INSERT INTO PROFILO (nome_profilo, nome_progetto)
VALUES (p_nome_profilo, p_nome_progetto);
    COMMIT;

```

```

END//


/*
* PROCEDURE: sp_profilo_delete
* PURPOSE: Rimozione di un profilo per un progetto software.
* USED BY: CREATORE
* NOTE: La rimozione del profilo comporta la rimozione (se esiste)
del PARTECIPANTE associato a quel profilo.
*
* @param IN p_nome_profilo - Nome del profilo da rimuovere
* @param IN p_nome_progetto - Nome del progetto software a cui
appartiene il profilo
* @param IN p_email_creatore - Email del creatore del progetto che
richiede la rimozione
*/
CREATE PROCEDURE sp_profilo_delete(
    IN p_nome_profilo VARCHAR(100),
    IN p_nome_progetto VARCHAR(100),
    IN p_email_creatore VARCHAR(100)
)
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN
            ROLLBACK;
            RESIGNAL;
        END;
    START TRANSACTION;
    -- CONTROLLI: Vedi la documentazione di questo check
    CALL sp_profilo_check(p_nome_profilo, p_email_creatore,
p_nome_progetto, FALSE);

    -- OK, rimuovo il profilo
    DELETE
    FROM PROFILO
    WHERE nome_profilo = p_nome_profilo
        AND nome_progetto = p_nome_progetto;
    COMMIT;
END//


/*
* PROCEDURE: sp_profilo_selectAllByProgetto
* PURPOSE: Restituisce la lista dei profili di un progetto software,
assieme alle competenze e ai livelli richiesti.
* USED BY: ALL
*
* @param IN p_nome_progetto - Nome del progetto software di cui si
vogliono ottenere i profili
*/
CREATE PROCEDURE sp_profilo_selectAllByProgetto(
    IN p_nome_progetto VARCHAR(100)

```

```

)
BEGIN
    START TRANSACTION;
    SELECT P.nome_profilo,
           SP.competenza,
           SP.livello_richiesto
      FROM PROFILO P
         LEFT JOIN SKILL_PROFILE SP ON P.nome_profilo =
SP.nome_profilo
        AND P.nome_progetto = SP.nome_progetto
   WHERE P.nome_progetto = p_nome_progetto
 ORDER BY P.nome_profilo, SP.competenza;
    COMMIT;
END//


/*
*  PROCEDURE: sp_profilo_nome_update
*  PURPOSE: Aggiornamento del nome di un profilo per un progetto software.
*  USED BY: CREATORE
*
*  @param IN p_nome_profilo - Nome del profilo da aggiornare
*  @param IN p_nome_progetto - Nome del progetto software a cui appartiene il profilo
*  @param IN p_nuovo_nome - Nuovo nome del profilo
*  @param IN p_email_creatore - Email del creatore del progetto che richiede l'aggiornamento
*/
CREATE PROCEDURE sp_profilo_nome_update(
    IN p_nome_profilo VARCHAR(100),
    IN p_nome_progetto VARCHAR(100),
    IN p_nuovo_nome VARCHAR(100),
    IN p_email_creatore VARCHAR(100)
)
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        RESIGNAL;
    END;
    START TRANSACTION;
    -- CONTROLLI: Vedi la documentazione di questo check
    CALL sp_profilo_check(p_nome_profilo, p_email_creatore,
p_nome_progetto, FALSE);

    -- OK, aggiorno il nome del profilo
    UPDATE PROFILO
       SET nome_profilo = p_nuovo_nome
      WHERE nome_profilo = p_nome_profilo
        AND nome_progetto = p_nome_progetto;

```

```

        COMMIT;
    END//


-- SKILL_PROFILO:
-- sp_skill_profilo_insert
-- sp_skill_profilo_delete
-- sp_skill_profilo_update
-- sp_skill_profilo_selectDiff

/*
* PROCEDURE: sp_skill_profilo_insert
* PURPOSE: Inserimento di una skill per un profilo di un progetto.
* USED BY: CREATORE
*
* @param IN p_nome_profilo - Nome del profilo del progetto software
per cui inserire la skill
* @param IN p_nome_progetto - Nome del progetto software a cui
appartiene il profilo
* @param IN p_email_creatore - Email del creatore del progetto che
richiede l'inserimento
* @param IN p_competenza - Competenza/skill richiesta per il profilo
del progetto da inserire
* @param IN p_livello_richiesto - Livello richiesto per la competenza
nel profilo del progetto (da 0 a 5)
*/
CREATE PROCEDURE sp_skill_profilo_insert(
    IN p_nome_profilo VARCHAR(100),
    IN p_nome_progetto VARCHAR(100),
    IN p_email_creatore VARCHAR(100),
    IN p_competenza VARCHAR(100),
    IN p_livello_richiesto TINYINT
)
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN
            ROLLBACK;
            RESIGNAL;
        END;
    START TRANSACTION;

    -- CONTROLLI: Vedi la documentazione di questo check
    CALL sp_skill_profilo_check(p_nome_profilo, p_email_creatore,
p_nome_progetto, p_competenza, TRUE);

    -- SIDE EFFECT: Rifiuto automaticamente le candidature che non
soddisfano il livello o la competenza richiesta
    UPDATE PARTECIPANTE P
    SET stato = 'rifiutato'
    WHERE P.nome_profilo = p_nome_profilo
        AND P.nome_progetto = p_nome_progetto

```

```

        AND P.statuto IN ('potenziale', 'accettato')
        AND NOT EXISTS (SELECT 1
                         FROM SKILL_CURRICULUM SC
                         WHERE SC.email_utente = P.email_utente
                               AND SC.competenza = p_competenza
                               AND SC.livello_effettivo >=
p_livello_richiesto);

-- OK, inserisco la skill nel profilo del progetto
INSERT INTO SKILL_PROFILE (nome_profilo, competenza,
nome_progetto, livello_richiesto)
VALUES (p_nome_profilo, p_competenza, p_nome_progetto,
p_livello_richiesto);
COMMIT;
END//


/*
* PROCEDURE: sp_skill_profilo_delete
* PURPOSE: Rimozione di una skill per un profilo di un progetto software.
* USED BY: CREATORE
* NOTE: La rimozione della skill comporta la rimozione (se esiste) del PARTECIPANTE associato a quel profilo e quella skill.
*
* @param IN p_nome_profilo - Nome del profilo del progetto software per cui rimuovere la skill
* @param IN p_nome_progetto - Nome del progetto software a cui appartiene il profilo
* @param IN p_competenza - Competenza/skill richiesta per il profilo del progetto da rimuovere
* @param IN p_email_creatore - Email del creatore del progetto che richiede la rimozione
*/
CREATE PROCEDURE sp_skill_profilo_delete(
    IN p_nome_profilo VARCHAR(100),
    IN p_nome_progetto VARCHAR(100),
    IN p_competenza VARCHAR(100),
    IN p_email_creatore VARCHAR(100)
)
BEGIN
DECLARE EXIT HANDLER FOR SQLEXCEPTION
BEGIN
    ROLLBACK;
    RESIGNAL;
END;
START TRANSACTION;

-- CONTROLLI: Vedi la documentazione di questo check
CALL sp_skill_profilo_check(p_nome_profilo, p_email_creatore,
p_nome_progetto, p_competenza, FALSE);

```

```

-- OK, rimuovo la skill dal profilo
DELETE
FROM SKILL_PROFIL0
WHERE nome_profilo = p_nome_profilo
    AND competenza = p_competenza
    AND nome_progetto = p_nome_progetto;
COMMIT;

END//


/*
* PROCEDURE: sp_skill_profilo_update
* PURPOSE: Aggiornamento del livello richiesto di una skill per un
profilo di un progetto software.
* USED BY: CREATORE
* NOTE: Se un PARTECIPANTE potenziale non soddisfa il nuovo livello
richiesto, la candidatura viene automaticamente rifiutata
*
* @param IN p_nome_profilo - Nome del profilo del progetto software
per cui aggiornare il livello richiesto della skill
* @param IN p_competenza - Competenza/skill richiesta per il profilo
del progetto da aggiornare
* @param IN p_nome_progetto - Nome del progetto software a cui
appartiene il profilo
* @param IN p_email_creatore - Email del creatore del progetto che
richiede l'aggiornamento
* @param IN p_nuovo_livello_richiesto - Nuovo livello richiesto per
la competenza nel profilo del progetto (da 0 a 5)
*/
CREATE PROCEDURE sp_skill_profilo_update(
    IN p_nome_profilo VARCHAR(100),
    IN p_competenza VARCHAR(100),
    IN p_nome_progetto VARCHAR(100),
    IN p_email_creatore VARCHAR(100),
    IN p_nuovo_livello_richiesto TINYINT
)
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN
            ROLLBACK;
            RESIGNAL;
        END;
    START TRANSACTION;

    -- CONTROLLI: Vedi la documentazione di questo check
    CALL sp_skill_profilo_check(p_nome_profilo, p_email_creatore,
p_nome_progetto, p_competenza, FALSE);

    -- SIDE EFFECT: Rifiuto automaticamente le candidature che non
soddisfano il nuovo livello richiesto

```

```

UPDATE PARTECIPANTE P
SET stato = 'rifiutato'
WHERE P.nome_profilo = p_nome_profilo
    AND P.nome_progetto = p_nome_progetto
    AND P.stato IN ('potenziale', 'accettato')
    AND NOT EXISTS (SELECT 1
                      FROM SKILL_CURRICULUM SC
                     WHERE SC.email_utente = P.email_utente
                           AND SC.competenza = p_competenza
                           AND SC.livello_effettivo >=
p_nuovo_livello_richiesto);

-- OK, aggiorno il livello richiesto della skill
UPDATE SKILL_PROFILo
SET livello_richiesto = p_nuovo_livello_richiesto
WHERE nome_profilo = p_nome_profilo
    AND competenza = p_competenza
    AND nome_progetto = p_nome_progetto;
COMMIT;
END//


/*
* PROCEDURE: sp_skill_profilo_selectDiff
* PURPOSE: Restituisce le competenze non presenti in un profilo di un
progetto software.
*
* @param IN p_nome_profilo - Nome del profilo del progetto software
* @param IN p_nome_progetto - Nome del progetto software a cui
appartiene il profilo
*/
CREATE PROCEDURE sp_skill_profilo_selectDiff(
    IN p_nome_profilo VARCHAR(100),
    IN p_nome_progetto VARCHAR(100)
)
BEGIN
    START TRANSACTION;
    SELECT S.competenza
    FROM SKILL S
    WHERE S.competenza NOT IN (SELECT SP.competenza
                               FROM SKILL_PROFILo SP
                               WHERE SP.nome_profilo = p_nome_profilo
                                     AND SP.nome_progetto =
p_nome_progetto);
    COMMIT;
END//


--- FOTO:
--- sp_foto_insert
--- sp_foto_delete
--- sp_foto_selectAll

```

```

/*
* PROCEDURE: sp_foto_insert
* PURPOSE: Inserimento di una foto per un progetto.
* USED BY: CREATORE
*
* @param IN p_nome_progetto - Nome del progetto a cui appartiene la
foto
* @param IN p_email_creatore - Email del creatore del progetto
* @param IN p_foto - Foto da inserire
*/
CREATE PROCEDURE sp_foto_insert(
    IN p_nome_progetto VARCHAR(100),
    IN p_email_creatore VARCHAR(100),
    IN p_foto MEDIUMBLOB
)
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN
            ROLLBACK;
            RESIGNAL;
        END;
    START TRANSACTION;
    -- CONTROLLI: Vedi la documentazione di questo check
    CALL sp_foto_check(p_nome_progetto, p_email_creatore, NULL, TRUE);

    -- OK, aggiungo la foto
    INSERT INTO FOTO (nome_progetto, foto)
    VALUES (p_nome_progetto, p_foto);
    COMMIT;
END//


/*
* PROCEDURE: sp_foto_delete
* PURPOSE: Rimozione di una foto per un progetto.
* USED BY: CREATORE
*
* @param IN p_nome_progetto - Nome del progetto a cui appartiene la
foto
* @param IN p_email_creatore - Email del creatore del progetto
* @param IN p_foto_id - ID della foto da rimuovere
*/
CREATE PROCEDURE sp_foto_delete(
    IN p_nome_progetto VARCHAR(100),
    IN p_email_creatore VARCHAR(100),
    IN p_foto_id INT
)
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN

```

```

        ROLLBACK;
        RESIGNAL;
    END;
    START TRANSACTION;
    -- CONTROLLI: Vedi la documentazione di questo check
    CALL sp_foto_check(p_nome_progetto, p_email_creatore, p_foto_id,
FALSE);

    -- OK, rimuovo la foto
    DELETE
    FROM FOTO
    WHERE nome_progetto = p_nome_progetto
        AND id = p_foto_id;
    COMMIT;
END//


/*
* PROCEDURE: sp_foto_selectAll
* PURPOSE: Visualizzazione di tutte le foto di un progetto.
* USED BY: ALL
*
* @param IN p_nome_progetto - Nome del progetto di cui visualizzare
le foto
*/
CREATE PROCEDURE sp_foto_selectAll(
    IN p_nome_progetto VARCHAR(100)
)
BEGIN
    START TRANSACTION;
    SELECT id, foto
    FROM FOTO
    WHERE nome_progetto = p_nome_progetto;
    COMMIT;
END//


DELIMITER ;

-- =====
-- VISTE
-- =====

-- Le 3 seguenti viste sono accessibili a tutti gli utenti autenticati
a livello di interfaccia, e non sono accessibili a utenti non
autenticati.
-- Sono visibili nella pagina statistiche.php ("Statistiche" nel
navbar).

-- Classifica dei top 3 utenti creatori, in base al loro valore di
affidabilità
CREATE OR REPLACE VIEW view_classifica_creatori_affidabilita AS

```

```

SELECT U.nickname
FROM CREATORE C
    JOIN UTENTE U ON C.email_utente = U.email
ORDER BY affidabilita DESC
LIMIT 3;

-- Stored procedure per visualizzare la classifica dei top 3 creatori
più affidabili
DELIMITER //
CREATE PROCEDURE view_classifica_creatori_affidabilita()
BEGIN
    SELECT * FROM view_classifica_creatori_affidabilita;
END//
DELIMITER ;

-- Classifica dei top 3 progetti APERTI che sono più vicini al proprio
completamento
CREATE OR REPLACE VIEW view_classifica_progetti_completamento AS
SELECT P.nome,
       P.budget,
       IFNULL((SELECT SUM(importo)
              FROM FINANZIAMENTO
             WHERE nome_progetto = P.nome), 0) AS tot_finanziamenti
FROM PROGETTO P
WHERE P.stato = 'aperto'
ORDER BY (tot_finanziamenti / budget) DESC
LIMIT 3;

-- Stored procedure per visualizzare la classifica dei top 3 progetti
più vicini al completamento
DELIMITER //
CREATE PROCEDURE view_classifica_progetti_completamento()
BEGIN
    SELECT * FROM view_classifica_progetti_completamento;
END//
DELIMITER ;

-- Classifica dei top 3 utenti, in base al TOTALE di finanziamenti
erogati
CREATE OR REPLACE VIEW view_classifica_utenti_finanziamento AS
SELECT U.nickname
FROM UTENTE U
    JOIN FINANZIAMENTO F ON U.email = F.email_utente
GROUP BY U.email
ORDER BY SUM(F.importo) DESC
LIMIT 3;

-- Stored procedure per visualizzare la classifica dei top 3 utenti
con più finanziamenti erogati
DELIMITER //

```

```

CREATE PROCEDURE view_classifica_utenti_finanziamento()
BEGIN
    SELECT * FROM view_classifica_utenti_finanziamento;
END// 
DELIMITER ; 

-- =====
-- TRIGGERS
-- =====

-- Similmente al blocco delle stored procedures, i trigger sono divisi
in base alla tabella di riferimento, con la seguente sintassi
generale...
-- NOME_TABELLA:
--   trg_nome_trigger

DELIMITER //

-- CREATORE:
--   trg_update_affidabilita_progetto
--   trg_update_affidabilita_finanziamento
--   trg_incrementa_progetti_creati

/*
*  TRIGGER: trg_update_affidabilita_progetto
*  PURPOSE: Aggiornare l'affidabilità di un creatore quando crea un
progetto
*/
CREATE TRIGGER trg_update_affidabilita_progetto
    AFTER INSERT
    ON PROGETTO
    FOR EACH ROW
BEGIN
    DECLARE tot_progetti INT;
    DECLARE progetti_finanziati INT;
    DECLARE new_aff DECIMAL(5, 2);

    -- Numero totale di progetti creati dal creatore
    SELECT COUNT(*)
    INTO tot_progetti
    FROM PROGETTO
    WHERE email_creatore = NEW.email_creatore;

    -- Numero di progetti del creatore che hanno ricevuto almeno un
finanziamento
    SELECT COUNT(DISTINCT P.nome)
    INTO progetti_finanziati
    FROM PROGETTO P
        JOIN FINANZIAMENTO F ON P.nome = F.nome_progetto
    WHERE P.email_creatore = NEW.email_creatore;

```

```

-- Calcolo la nuova affidabilità del creatore
IF tot_progetti > 0 THEN
    SET new_aff = (progetti_finanziati / tot_progetti) * 100;
ELSE
    SET new_aff = 0;
END IF;

UPDATE CREATORE
SET affidabilita = new_aff
WHERE email_utente = NEW.email_creatore;
END//


/*
* TRIGGER: trg_update_affidabilita_finanziamento
* PURPOSE: Aggiornare l'affidabilità di un creatore quando un
progetto da lui creato viene finanziato
*/
CREATE TRIGGER trg_update_affidabilita_finanziamento
AFTER INSERT
ON FINANZIAMENTO
FOR EACH ROW
BEGIN
    DECLARE email VARCHAR(100);
    DECLARE tot_progetti INT;
    DECLARE progetti_finanziati INT;
    DECLARE new_aff DECIMAL(5, 2);

    -- Recupero l'email del creatore del progetto
    SELECT email_creatore
    INTO email
    FROM PROGETTO
    WHERE nome = NEW.nome_progetto;

    -- Numero totale di progetti creati dal creatore
    SELECT COUNT(*)
    INTO tot_progetti
    FROM PROGETTO
    WHERE email_creatore = email;

    -- Numero di progetti del creatore che hanno ricevuto almeno un
finanziamento
    SELECT COUNT(DISTINCT P.nome)
    INTO progetti_finanziati
    FROM PROGETTO P
        JOIN FINANZIAMENTO F ON P.nome = F.nome_progetto
    WHERE P.email_creatore = email;

    -- Calcolo la nuova affidabilità del creatore
    IF tot_progetti > 0 THEN

```

```

        SET new_aff = (progetti_finanziati / tot_progetti) * 100;
    ELSE
        SET new_aff = 0;
    END IF;

    UPDATE CREATORE
    SET affidabilita = new_aff
    WHERE email_utente = email;
END//


/*
* TRIGGER: trg_incrementa_progetti_creati
* PURPOSE: Aumentare il numero di progetti creati da un creatore
*/
CREATE TRIGGER trg_incrementa_progetti_creati
    AFTER INSERT
    ON PROGETTO
    FOR EACH ROW
BEGIN
    UPDATE CREATORE
    SET nr_progetti = nr_progetti + 1
    WHERE email_utente = NEW.email_creatore;
END//


-- PROGETTO:
--   trg_update_stato_progetto

/*
* TRIGGER: trg_update_stato_progetto
* PURPOSE: Cambia lo stato di un progetto in 'chiuso' quando il
budget è stato raggiunto e rifiuta le candidature pendenti.
*/
CREATE TRIGGER trg_update_stato_progetto
    AFTER INSERT
    ON FINANZIAMENTO
    FOR EACH ROW
BEGIN
    DECLARE tot_finanziamento DECIMAL(10, 2);
    DECLARE budget_progetto DECIMAL(10, 2);

    -- Calcola il budget del progetto
    SELECT budget
    INTO budget_progetto
    FROM PROGETTO
    WHERE nome = NEW.nome_progetto;

    -- Calcola il totale del finanziamento per il progetto
    SELECT SUM(importo)
    INTO tot_finanziamento
    FROM FINANZIAMENTO

```

```

WHERE nome_progetto = NEW.nome_progetto;

-- Se il totale del finanziamento è >= al budget del progetto,
allora lo stato del progetto diventa 'chiuso'
IF tot_finanziamento >= budget_progetto THEN
    -- Chiudo il progetto
    UPDATE PROGETTO
    SET stato = 'chiuso'
    WHERE nome = NEW.nome_progetto;

    -- Rifiuto tutte le candidature pendenti
    UPDATE PARTECIPANTE
    SET stato = 'rifiutato'
    WHERE nome_progetto = NEW.nome_progetto
        AND stato = 'potenziale';
END IF;
END//


DELIMITER ;

-- =====
-- EVENTI
-- =====

DELIMITER //


/*
* EVENTO: ev_chiudi_progetti_scaduti
* PURPOSE: Chiudere automaticamente i progetti scaduti, eseguito ogni
giorno
*/
CREATE EVENT ev_chiudi_progetti_scaduti
    ON SCHEDULE EVERY 1 DAY
    DO
    BEGIN
        UPDATE PROGETTO
        SET stato = 'chiuso'
        WHERE stato = 'aperto'
            AND data_limite < CURDATE();
    END//


DELIMITER ;

```