



891923 · Ramolivaz André
892614 · Tomasin Alberto
892539 · Dinato Simone

EUResearchHub: Progresso in sinergia

Report per il progetto di CT0006 - Basi di Dati



Ca' Foscari
University
of Venice

Indice



| | | |
|----------|--|-----------|
| 1 | Introduzione | 4 |
| 1.1 | Contesto dell'applicazione | 5 |
| 1.2 | Funzionalità principali | 5 |
| 1.2.1 | Gestione dei progetti di ricerca | 5 |
| 1.2.2 | Processo di valutazione e storico modifiche | 6 |
| 1.2.3 | Interazione tra ricercatori e valutatori | 6 |
| 1.3 | Obiettivi e scopo del documento | 6 |
| 2 | Progettazione della base di dati | 7 |
| 2.1 | Raccolta ed analisi dei requisiti | 8 |
| 2.1.1 | Quadro generale | 8 |
| 2.1.2 | Vincoli preliminari | 8 |
| 2.1.3 | Ruoli | 9 |
| 2.2 | Progettazione concettuale | 9 |
| 2.2.1 | Schema ad oggetti | 9 |
| 2.2.2 | Giustificazione delle decisioni progettuali | 11 |
| 2.3 | Progettazione logica | 11 |
| 2.3.1 | Schema logico | 11 |
| 2.3.2 | Giustificazione delle decisioni progettuali | 13 |
| 2.4 | Progettazione fisica | 13 |
| 2.4.1 | Istruzioni SQL | 13 |
| 2.4.2 | Specifiche integrità dei dati | 13 |
| 2.4.3 | Gestione dei ruoli e politiche di autorizzazione | 16 |
| 2.4.4 | Performance: indici e materialized view | 18 |
| 3 | Sviluppo back-end | 20 |
| 3.1 | Scelta di Flask come framework | 21 |
| 3.2 | Architettura dell'applicazione | 21 |
| 3.2.1 | Ruolo del back-end nell'architettura complessiva | 21 |
| 3.2.2 | Organizzazione delle directory e dei file del progetto | 21 |
| 3.2.3 | Principali componenti del back-end | 22 |
| 3.3 | Interazione con il DBMS PostgreSQL | 22 |
| 3.3.1 | Utilizzo di SQLAlchemy per l'interazione con il database | 23 |
| 3.3.2 | Esecuzione delle query principali e gestione dei risultati | 23 |
| 3.4 | Sicurezza del back-end | 25 |
| 3.4.1 | Utilizzo di Flask-WTF per gestire i moduli e i token CSRF | 25 |
| 3.4.2 | Meccanismi di autenticazione e autorizzazione | 25 |
| 3.4.3 | Salting e hashing delle password degli utenti | 25 |
| 3.4.4 | Protezione da SQL Injection | 25 |
| 4 | Implementazione front-end | 26 |
| 4.1 | Design finale | 27 |
| 4.2 | Integrazione tra front-end e back-end | 28 |
| 5 | Contributo al progetto | 30 |
| 5.1 | Processo di sviluppo | 31 |
| 5.2 | Analitiche Github | 31 |

Capitolo 1

Introduzione



Questo documento descrive come abbiamo progettato e implementato EUResearchHub: una web application user-friendly per rendere possibile la valutazione interna dei progetti di ricerca proposti per il finanziamento da parte dell'Unione Europea per il progetto accademico del corso: *Basi di Dati [CT0006] - Ca' Foscari Università di Venezia*.

 *Se vogliamo avere idee originali, dobbiamo investire nella ricerca. Non si tratta solo di investire nel settore privato, ma anche nel settore pubblico · Elon Musk* 

1.1 Contesto dell'applicazione

EUResearchHub è un sistema che mira a rendere efficiente la gestione e la valutazione dei progetti di ricerca, offrendo una piattaforma accattivante e funzionale sia per i ricercatori che per i valutatori. Abbiamo scelto questo tema principalmente per la sua rilevanza e attualità nel panorama della ricerca e dell'istruzione superiore. Nel corso degli anni, il contesto della ricerca scientifica è cambiato notevolmente. Con l'avvento dell'era digitale, la ricerca è diventata sempre più interconnessa e collaborativa, permettendo ai ricercatori di tutto il mondo di lavorare insieme per risolvere problemi complessi. Tuttavia, questo cambiamento ha anche portato ad un aumento della competizione per i finanziamenti di ricerca, soprattutto da parte dell'Unione Europea, che è uno dei principali finanziatori di progetti di ricerca a livello globale.

Negli ultimi anni, l'Unione Europea ha lanciato diversi programmi di finanziamento, come Orizzonte 2020 e il suo successore, Orizzonte Europa, che hanno l'obiettivo di sostenere la ricerca e l'innovazione in diversi settori, come la salute, l'energia, l'ambiente e l'istruzione. Per accedere a questi fondi, le istituzioni di ricerca devono sottoporre i propri progetti a processi di valutazione rigorosi e competitivi.

In questo contesto, è fondamentale che le istituzioni di ricerca dispongano di strumenti efficaci per gestire e valutare i progetti di ricerca internamente, al fine di aumentare le possibilità di ottenere finanziamenti. EUResearchHub è stata concepita proprio per rispondere a questa necessità, fornendo una piattaforma che facilita la comunicazione e la collaborazione tra ricercatori e valutatori, permettendo loro di lavorare insieme in modo efficiente per migliorare la qualità dei progetti di ricerca proposti.

1.2 Funzionalità principali

Nel progettare EUResearchHub, abbiamo posto particolare enfasi sull'offerta di funzionalità che consentano una gestione efficiente e trasparente dei progetti di ricerca, facilitando il processo di valutazione e promuovendo una comunicazione efficace tra ricercatori e valutatori tramite un sistema di messaggistica e storico dei documenti.

A tal proposito è importante definire fin da subito definire due grandi aree. Una è quella riguardante il ricercatore e tutte le sue funzionalità mentre l'altra è legata al valutatore.

In questa sezione, ci concentreremo sulle tre funzionalità chiave per ogni area: la gestione dei progetti di ricerca, il processo di valutazione, lo storico delle modifiche e l'interazione tra ricercatori e valutatori.

1.2.1 Gestione dei progetti di ricerca

La gestione dei progetti di ricerca è una componente fondamentale per EUResearchHub. Il sistema è stato progettato per consentire ai ricercatori di creare e visualizzare facilmente nuovi progetti, inserendo informazioni pertinenti, come il titolo e la descrizione del progetto o aggiungere anche partecipanti al progetto. Una volta creato il progetto, i ricercatori possono anche caricare documenti in formato PDF, come il piano di gestione dei dati, i materiali etici che verranno sottoposti ai valutatori durante processo di valutazione. I valutatori quindi non potranno aggiungere utenti al progetto o creare nuovi progetti e caricare documenti per gli stessi.

1.2.2 Processo di valutazione e storico modifiche

Una volta che i progetti di ricerca sono stati creati e sottomessi per la valutazione, i valutatori possono accedervi per valutarli. Il processo di valutazione è stato progettato per essere il più trasparente e approfondito ma allo stesso minimal possibile. I valutatori possono scaricare i documenti relativi al progetto, leggerli, analizzarli, e commentarli quindi creare un report di valutazione per ogni documento e caricarlo nuovamente sulla piattaforma per renderlo visibile ai ricercatori interessati. Questo report può includere commenti generali sul progetto, suggerimenti per miglioramenti e valutazioni specifiche per ogni documento allegato.

Una caratteristica importante del sistema è la possibilità di tracciare lo storico delle modifiche dei documenti e quindi dei progetti sottomessi. Ogni volta che un ricercatore apporta modifiche ad un documento e lo carica nel sistema EUResearchHub registra una nuova versione, mantenendo una copia della versione precedente. Questo permette sia ai ricercatori che ai valutatori di monitorare l'evoluzione del progetto nel tempo e di comprendere come le modifiche apportate abbiano influito sulla valutazione finale del progetto.

Lo storico delle modifiche è particolarmente utile nei casi in cui i valutatori richiedono modifiche ai progetti prima di approvarli. In questi casi, i ricercatori possono utilizzare lo storico delle versioni per confrontare le diverse versioni del progetto e identificare le aree in cui è necessario apportare miglioramenti. Per quanto concerne il processo di valutazione, lo status potrà assumere diversi valori (submitted for evaluation, required changes ...) ma questi valori possono cambiare solo se tutti i documenti inseriti dal ricercatore sono stati valutati. Entrambi gli utenti possono visionare la percentuale dei documenti valutati in corso e indicativamente quanti ne mancano tramite l'evaluation progress bar.

1.2.3 Interazione tra ricercatori e valutatori

Insieme allo storico dei documenti la comunicazione tra ricercatori e valutatori è un aspetto cruciale per garantire un processo di valutazione efficace e monitorare il ciclo di vita di un progetto. EUResearchHub è dotata di una componente di messaggistica per ogni progetto che consente ai ricercatori e ai valutatori di interagire direttamente all'interno della piattaforma. Questa funzionalità consente ai ricercatori di richiedere chiarimenti sui report di valutazione, suggerimenti su come migliorare i loro progetti e informazioni aggiuntive sui requisiti specifici dei finanziamenti. Allo stesso tempo, i valutatori possono rispondere alle domande dei ricercatori in modo anonimo, garantendo l'obiettività e l'imparzialità del processo di valutazione.

I valutatori possono utilizzare questa funzione per fornire feedback specifico e mirato sui documenti, ad esempio evidenziando passaggi che richiedono chiarimenti o suggerendo modifiche al testo. Questi messaggi possono essere utilizzati dai ricercatori per apportare miglioramenti ai loro progetti e dai valutatori per fare riferimento a punti specifici nei loro report di valutazione (ad esempio, "si veda la nota a pagina 4") e possono essere utilizzati anche come strumento organizzativo/manageriale per i ricercatori del progetto.

1.3 Obiettivi e scopo del documento

In questo documento, si spiega come abbiamo progettato e modellato la base di dati partendo dalla raccolta ed analisi dei requisiti fino alla progettazione fisica con particolare enfasi sul controllo degli accessi e integrità dei dati per poi passare a descrivere l'implementazione del back-end e del front-end ed infine concludere con una panoramica generale del processo di sviluppo e il contributo individuale dei membri del gruppo di lavoro.

Capitolo 2

Progettazione della base di dati



L'efficacia di EURsearchHub per la valutazione interna dei progetti di ricerca dipende in gran parte dalla qualità e dall'efficienza della base di dati che ne supporta il funzionamento. In questo capitolo, ci concentreremo sulla progettazione e modellazione della base di dati, analizzando le varie fasi del processo, tra cui la raccolta e l'analisi dei requisiti, la progettazione concettuale, logica e fisica, e le considerazioni relative all'integrità dei dati, al controllo degli accessi e alle performance del sistema.

2.1 Raccolta ed analisi dei requisiti

La fase di raccolta ed analisi dei requisiti è stato uno dei più importanti pilastri per comprendere e definire le specifiche del sistema. In questa fase embrionale, abbiamo identificato le esigenze degli utenti e le funzionalità che la base di dati deve supportare per garantire il corretto funzionamento di tutto il sistema.

2.1.1 Quadro generale

Come già accennato precedentemente EURsearchHub ha il compito di fornire una piattaforma di interazione tra ricercatori e valutatori per la valutazione interna dei progetti di ricerca. Il sistema deve quindi essere in grado di gestire progetti, documenti, messaggi e report di valutazione, oltre a fornire supporto per l'autenticazione e l'autorizzazione degli utenti.

Quindi, analizzando attentamente il problema, le entità chiave individuate sono:

- `project` : contiene informazioni sui progetti di ricerca, come il titolo, lo stato (approvato, sottomesso per valutazione, richiede modifiche, non approvato), la descrizione e la data di creazione.
- `evaluation_windows` : rappresenta le finestre temporali in cui i progetti possono essere valutati, con date di inizio e fine.
- `messages` : contiene i messaggi scambiati tra ricercatori e valutatori riguardo ai progetti.
- `documents` e `document_versions` : gestiscono i documenti associati ai progetti e le loro diverse versioni.
- `document_types` : classifica i diversi tipi di documenti utilizzati nel sistema. Si ipotizza che vi siano dei tipi di documento pre-stabiliti dall'ente che saranno poi i soli tipi di documenti che un ricercatore potrà andare a caricare.
- `users` : `evaluators` e `researchers` che contengono informazioni sugli utenti del sistema, come nome, cognome, password ed e-mail.
- `evaluation_reports` : archivia i report di valutazione generati dai valutatori per i documenti dei progetti di ricerca.

2.1.2 Vincoli preliminari

Definite le entità chiave risulta fondamentale inserire dei vincoli per garantire l'integrità e coerenza dei dati che andremo ad inserire. Nel nostro contesto, assicurarsi che i dati siano accurati e coerenti è fondamentale per evitare errori nella valutazione e facilitare il processo decisionale. Più in dettaglio i vincoli necessari individuati in questa prima fase sono stati:

- Vincolo sulla tabella `evaluation_windows` : necessario per assicurarsi che la data di inizio sia sempre inferiore o uguale alla data di fine. Questo vincolo permette di prevenire la creazione di finestre di valutazione con date illogiche o errate, garantendo che le date siano coerenti e rispettino un ordine temporale corretto.
- Vincoli sulla lunghezza delle password per le due tipologie di users `researchers` ed `evaluators` : necessari a garantire che le password degli utenti abbiano una lunghezza minima di 8 caratteri. Questo requisito aumenta la sicurezza degli account, riducendo la probabilità di accessi non autorizzati attraverso attacchi brute-force o altre tecniche di hacking. Sebbene questo controllo possa essere fatto anche dal back-end implementarlo anche nella base di dati permette di centralizzare il vincolo.

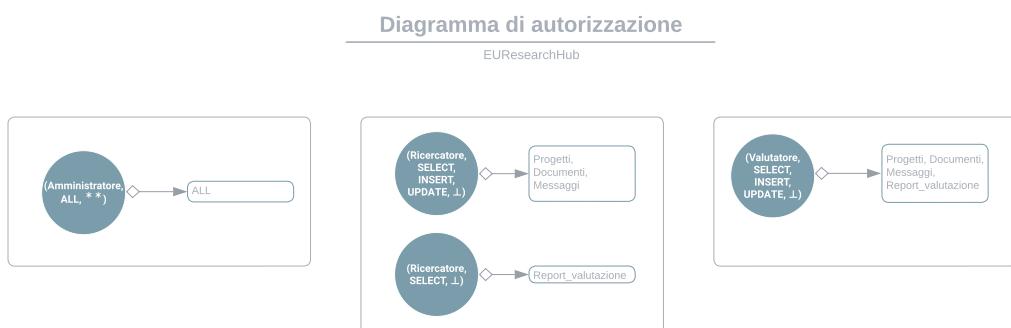
- Vincoli sull'email per **researchers** ed **evaluators**: necessari per assicurarsi che gli indirizzi email siano unici, validi e rispettino il formato standard delle email. Questo vincolo previene l'inserimento di indirizzi email non validi o errati, garantendo una corretta comunicazione tra i ricercatori e i valutatori.

2.1.3 Ruoli

Un altro step necessario per proseguire con la raccolta ed analisi dei requisiti è individuare i ruoli e le politiche di autorizzazione. Questo è fondamentale per garantire la sicurezza e la corretta separazione delle responsabilità all'interno di una web app. Nel caso di EUResearchHub, abbiamo definito tre ruoli principali: **Admin**, **Researcher** e **Evaluator**. Ognuno di questi ruoli dovrà avere privilegi e restrizioni specifici per garantire che gli utenti possano accedere e modificare solo le informazioni pertinenti alle loro funzioni.

Implementare ruoli ben definiti e separati ci permette di ottenere una maggiore sicurezza, una riduzione del rischio di abusi o accessi non autorizzati e una migliore organizzazione delle responsabilità all'interno del nostro sistema.

Da una prima analisi del problema otteniamo il seguente diagramma di autorizzazioni:



Nel contesto descritto, non sembra necessario inserire ereditarietà o delega di permessi tra i ruoli, motivo per cui nel diagramma soprafigurante i nodi non sono legati tra di essi. Ogni ruolo ha responsabilità e permessi molto specifici e distinti, e non sembra esserci un caso d'uso in cui un ruolo avrebbe bisogno di ereditare i permessi di un altro ruolo.

2.2 Progettazione concettuale

Dopo l'analisi e raccolta dei requisiti il secondo pilastro necessario per la progettazione e modellazione del nostro database è la progettazione concettuale. Essa consente di strutturare gli elementi chiave del sistema prima identificati. Nel nostro caso, è stato utilizzato Lucidchart per sviluppare un modello ad oggetti che rappresenta le tabelle, le relazioni, i vincoli e le invarianti, fornendo una panoramica comprensiva e astratta dell'architettura del database.

Nel modello concettuale, le tabelle rappresentano le entità principali del sistema, come i ricercatori, i valutatori, i progetti, i documenti e così via. Le relazioni tra queste entità sono state modellate con parzialità e molteplicità, indicando come le diverse entità interagiscono tra loro e quali sono le dipendenze tra di esse.

Gli invarianti sono stati evidenziati infondo alle tabelle interessate per garantire l'integrità e la consistenza dei dati. Ad esempio, alcuni dei vincoli menzionati in precedenza, come il controllo sulla lunghezza delle password e il formato degli indirizzi email, sono stati inclusi nel modello per assicurare che i dati inseriti nel sistema rispettino le regole e standard.

2.2.1 Schema ad oggetti

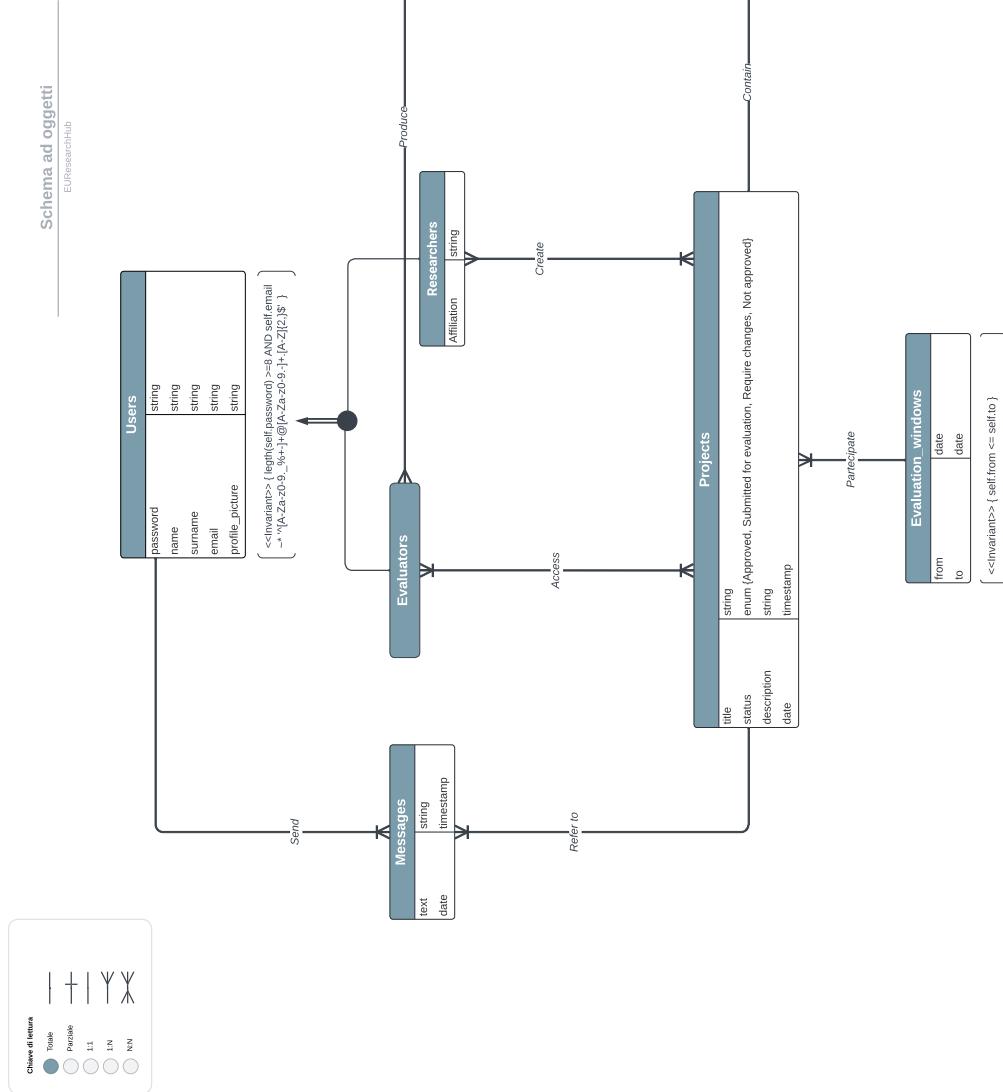


Figura 2.1: Schema ad oggetti

2.2.2 Giustificazione delle decisioni progettuali

Di seguito sono riportate le giustificazioni delle principali decisioni progettuali per garantire che il sistema fosse scalabile, efficiente e facile da mantenere:

- **Utilizzo di tipi enum:** Per rappresentare lo status dei progetti, creeremo un tipo enumerato `enum_status`. Questo garantisce che solo i valori predefiniti possano essere inseriti nella colonna status della tabella `projects`, garantendo la consistenza dei dati e facilitando le query sullo stato dei progetti.
- **Definita una relazione di sottoclasse:** Sono state definite due sottoclassi partizione `researchers` e `evaluators` che ereditano gli attributi da `users` per evitare eventuali ridondanze e permettere durante la progettazione logica di effettuare l'implementazione corretta.
- **Suddivisione delle informazioni in tabelle separate:** Al fine di garantire la normalizzazione del database e ridurre la ridondanza dei dati, le informazioni sono state suddivise in tabelle separate. Ad esempio, le informazioni sui documenti e le loro versioni sono state divise nelle tabelle `documents` e `document_versions`, permettendo di gestire più versioni di uno stesso documento in modo efficiente. In maniera analoga anche le `evaluation_windows` vengono salvate in una tabella separata da `projects`.
- **Utilizzo di invarianti:** già accennate in precedenza.

2.3 Progettazione logica

Ora analizziamo la fase intermedia della nostra modellazione: La progettazione logica. In questa fase, abbiamo tradotto il modello concettuale ad oggetti in uno schema logico che è stato utilizzato poi durante la creazione del database. La progettazione logica è essenziale per diverse ragioni:

1. Consente di identificare e risolvere eventuali problemi nella struttura dei dati prima di passare alla progettazione fisica.
2. Semplifica la mappatura tra il modello concettuale e il modello fisico, riducendo il rischio di errori e migliorando la coerenza dei dati.
3. Fornisce un'opportunità per ottimizzare le prestazioni del database, ad esempio selezionando gli indici appropriati e creando viste materializzate per migliorare l'efficienza delle query che si andranno ad implementare durante la progettazione fisica.

2.3.1 Schema logico

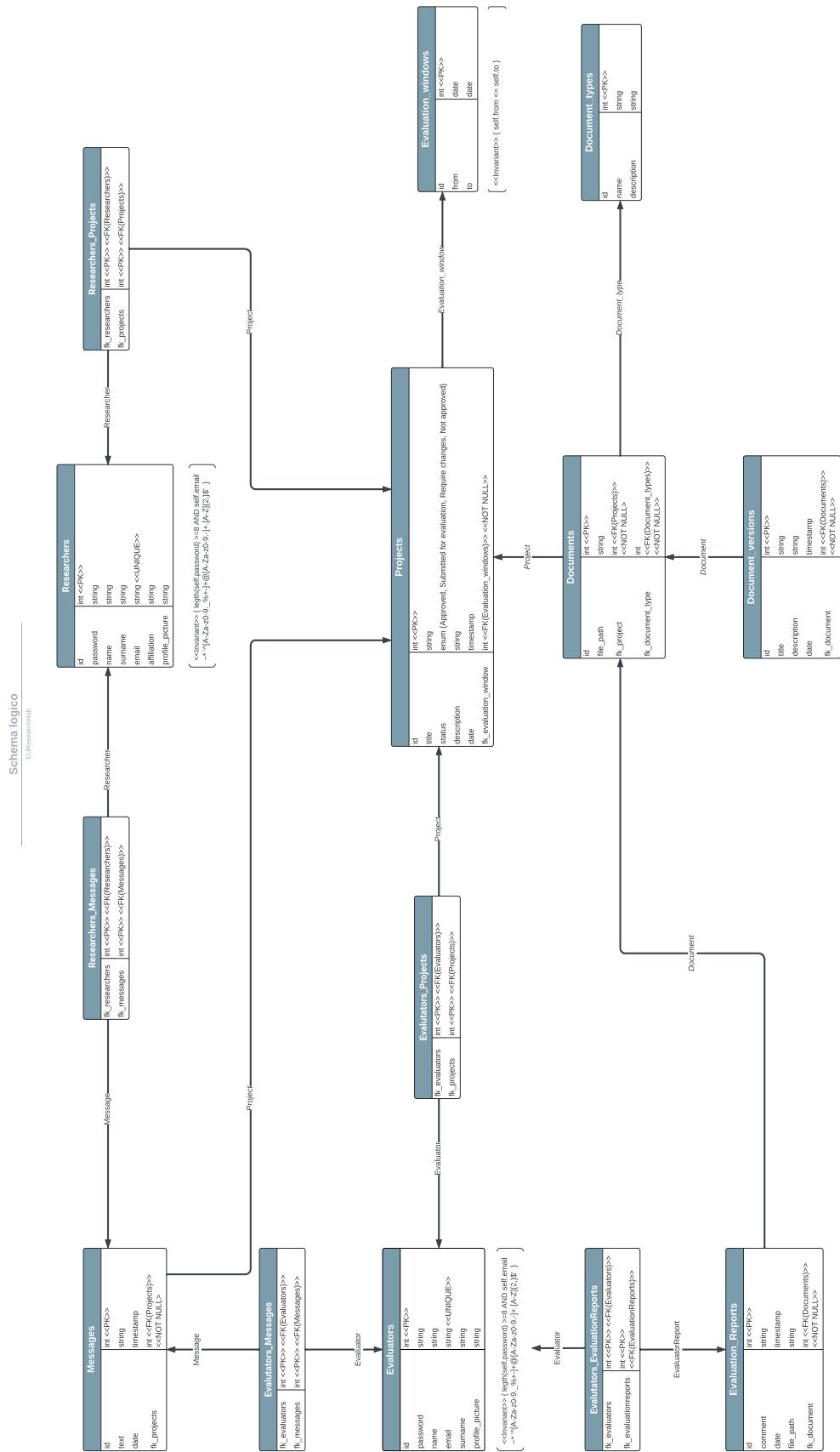


Figura 2.2: Schema logico

2.3.2 Giustificazione delle decisioni progettuali

Di seguito sono riportate le giustificazioni di alcune delle decisioni chiave per questa fase:

- **Tradotta la relazione di sottoclasse tramite partizionamento orizzontale:** Dalla gerarchia iniziale si è arrivati a suddividere le due sottoclassi partizione `researchers` e `evaluators` in due tabelle a se stanti che ereditano gli attributi da `users` poiché non si voleva complicare maggiormente la vista di tutti gli elementi della super classe tramite partizionamento verticale e neanche generalizzare troppo utilizzando una discriminante e raggruppando tutti gli attributi nella tabella unica `users`.
- **Chiavi esterne e relazioni:** Sono state utilizzate chiavi esterne per stabilire relazioni tra le diverse tabelle, garantendo l'integrità referenziale e semplificando le query tra tabelle correlate. Ad esempio, la tabella `documents` ha una chiave esterna che fa riferimento alla tabella `projects`, indicando a quale progetto appartiene ciascun documento.
- **Utilizzo di tabelle intermedie per relazioni multi-a-molti:** Per gestire le relazioni multi-a-molti tra entità, sono state create tabelle intermedie, come `researchers_projects` e `evaluators_messages`. Queste tabelle consentono di associare più ricercatori a più progetti e più messaggi a più valutatori, garantendo la flessibilità del sistema e semplificando le query tra tabelle correlate.
- **Inserito id univoco per ogni tabella:** Ogni tabella è caratterizzata da un ID univoco sotto forma di intero auto incrementato.
- **Inserito vincolo UNIQUE sulle mail:** Ogni utente è caratterizzata da un email che deve essere univoco.

2.4 Progettazione fisica

La progettazione fisica presenta l'ultima stazione per completare questo capitolo. Vista l'analisi a 360° del database, e di come dovrebbe funzionare l'applicativo non resta che applicare le osservazioni fatte su DBeaver. I costrutti per la creazione si basano sul linguaggio PostgreSQL.

2.4.1 Istruzioni SQL

Per le istruzioni SQL di creazione della base di dati si veda allegato `dump.sql`.

2.4.2 Specifiche integrità dei dati

Definite le tabelle in SQL chiave risulta fondamentale implementare i vincoli individuati durante la raccolta ed analisi dei requisiti e aggiungerne nuovi individuati durante la progettazione. Più in dettaglio, i vincoli definitivi implementati in PosgreSQL sono stati:

Vincolo sulla tabella `evaluation_windows`

```
1 ALTER TABLE "EUREsearchHub".evaluation_windows
2 ADD CONSTRAINT evaluation_windows_dates_check CHECK ("from" <= "to");
```

Vincoli sulla lunghezza delle password per le due tipologie di users `researchers` ed `evaluators`

```
1 ALTER TABLE "EUREsearchHub".researchers
2 ADD CONSTRAINT researchers_password_length_check CHECK (LENGTH("password") >= 8);
3
4 ALTER TABLE "EUREsearchHub".evaluators
5 ADD CONSTRAINT evaluators_password_length_check CHECK (LENGTH("password") >= 8);
```

Vincoli sull'email per `researchers` ed `evaluators`:

```

1 ALTER TABLE "EUResearchHub".researchers
2 ADD CONSTRAINT researchers_email_check CHECK (email ~*
→ '^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z]{2,}$');
3
4 ALTER TABLE "EUResearchHub".evaluators
5 ADD CONSTRAINT researchers_email_check CHECK (email ~*
→ '^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z]{2,}$');

```

L'utilità del vincolo è già stata spiegata ma risulta interessante, in questo caso, spigarne i tecnicismi. Il vincolo utilizza una regex (espressione regolare). Per comprenderne meglio il funzionamento possiamo dividere la stringa in 4 parti:

1. **[A-Za-z0-9._%+-]+**: questa parte dell'espressione indica che l'indirizzo email deve iniziare (^) con una sequenza di almeno un carattere (+) che può essere una lettera maiuscola o minuscola, un numero, un punto, un underscore, un simbolo di percentuale, un simbolo più o un simbolo meno.
2. **@**: questo simbolo indica che, dopo la sequenza iniziale di caratteri, deve essere presente il simbolo "@" che separa la parte locale dell'indirizzo email dal dominio.
3. **[A-Za-z0-9.-]+**: questa parte dell'espressione indica che, dopo il simbolo "@", deve esserci una sequenza di almeno un carattere (+) che può essere una lettera maiuscola o minuscola, un numero, un punto o un trattino. Questa sequenza rappresenta il dominio dell'indirizzo email (ad esempio, "gmail.com" o "example.org").
4. **[A-Z]2,\$**: questa parte finale dell'espressione indica che, dopo il dominio, deve essere presente un punto seguito da almeno due lettere maiuscole o minuscole (2.). Questa sequenza rappresenta il suffisso del dominio, come ".com", ".org" o ".net". Il simbolo \$ alla fine dell'espressione indica che la stringa deve terminare con il suffisso del dominio.

Trigger sulla tabella `document_versions`:

```

1 CREATE FUNCTION check_document_version_date()
2 RETURNS TRIGGER AS $$$
3 DECLARE
4     previous_version_date TIMESTAMP;
5 BEGIN
6     SELECT "date" INTO previous_version_date
7     FROM "EUResearchHub".document_versions
8     WHERE fk_document = NEW.fk_document
9     ORDER BY "date" DESC
10    LIMIT 1;
11
12    IF previous_version_date IS NOT NULL AND previous_version_date >= NEW."date" THEN
13        RAISE EXCEPTION 'New document version date must be greater than the previous
→ version date';
14    END IF;
15
16    RETURN NEW;
17 END;
18 $$ LANGUAGE plpgsql;
19
20 CREATE TRIGGER check_document_versions_date
21     BEFORE INSERT ON "EUResearchHub".document_versions
22     FOR EACH ROW
23     EXECUTE FUNCTION check_document_version_date();

```

Il trigger `check_document_versions_date` è stato creato per verificare che la data della nuova versione di un documento (NEW.date) sia sempre maggiore rispetto alla data dell'ultima versione registrata nella base di dati.

La funzione `check_document_version_date()` viene eseguita prima dell'inserimento di una nuova versione del documento nella tabella `document_versions`. Se la condizione non è soddisfatta, viene generato un errore e l'operazione di inserimento viene interrotta. Questo trigger garantisce che le versioni dei documenti siano sempre ordinate in modo cronologico, facilitando la gestione delle modifiche e delle revisioni dei documenti.

Trigger sulla tabella `projects`:

```

1  CREATE FUNCTION check_project_status_change()
2  RETURNS TRIGGER AS $$ 
3  DECLARE
4      document_count INTEGER;
5      evaluated_document_count INTEGER;
6  BEGIN
7      SELECT COUNT(*) INTO document_count
8      FROM "EUREsearchHub".documents
9      WHERE fk_project = NEW.id;
10
11     SELECT COUNT(DISTINCT d.id) INTO evaluated_document_count
12     FROM "EUREsearchHub".documents d
13     JOIN "EUREsearchHub".evaluation_reports er ON d.id = er.fk_document
14     WHERE d.fk_project = NEW.id;
15
16     IF document_count <> evaluated_document_count THEN
17         RAISE EXCEPTION 'Cannot change the status of a project if not all documents have
18         been evaluated';
19     END IF;
20
21     RETURN NEW;
22 END;
23 $$ LANGUAGE plpgsql;
24
25 CREATE TRIGGER check_projects_status_change
26     BEFORE UPDATE ON "EUREsearchHub".projects
27     FOR EACH ROW
28     WHEN (OLD.status <> NEW.status AND NEW.status IN ('approved', 'require changes',
29         'not approved'))
30     EXECUTE FUNCTION check_project_status_change();

```

Il trigger `check_projects_status_change` è stato sviluppato per garantire che lo stato di un progetto possa essere modificato solo se tutti i documenti associati al progetto sono stati valutati. La funzione associata viene eseguita prima dell'aggiornamento dello stato del progetto nella tabella `projects`. Se la condizione non è soddisfatta, viene generato un errore e l'operazione di aggiornamento viene interrotta. Questo trigger assicura che i progetti non vengano approvati, richiedano modifiche o siano respinti finché non vengono completamente valutati, garantendo un processo di valutazione accurato e completo.

Trigger sulle tabelle `researchers` e `evaluators`:

```

1 CREATE FUNCTION check_unique_email()
2   RETURNS TRIGGER AS
3   $$
4 BEGIN
5   IF EXISTS (SELECT 1 FROM "EUResearchHub".researchers WHERE email = NEW.email) THEN
6     RAISE EXCEPTION 'Another user is usinng this email, please change it';
7   END IF;
8
9   IF EXISTS (SELECT 1 FROM "EUResearchHub".evaluators WHERE email = NEW.email AND id
10    → <> NEW.id) THEN
11     RAISE EXCEPTION 'Another user is usinng this email, please change it';
12   END IF;
13
14   RETURN NEW;
15 END;
16 $$;
17 LANGUAGE plpgsql;
18
19 CREATE TRIGGER check_unique_email_insert_update
20 BEFORE INSERT OR UPDATE ON "EUResearchHub".evaluators
21 FOR EACH ROW
22 EXECUTE FUNCTION check_unique_email();
23 CREATE TRIGGER check_unique_email_insert_update
24 BEFORE INSERT OR UPDATE ON "EUResearchHub".researchers
25 FOR EACH ROW
26 EXECUTE FUNCTION check_unique_email();

```

Il trigger `check_unique_email` viene utilizzato per garantire che le email siano uniche tra le tabelle `researchers` e `evaluators`. Ciò significa che un ricercatore non può essere anche un valutatore, in quanto il loro indirizzo email deve essere univoco in entrambe le tabelle. Questo trigger, oltre a permettere di far rispettare la scelta della tipologia gerarchica di `users` presa durante la progettazione concettuale risulta utile anche perché l'immagine del profilo durante lo sviluppo verrà salvata salvate come `email.pdf`. La funzione associata al trigger esegue i seguenti controlli:

- Verifica se l'email inserita esiste già nella tabella `researchers`. Se l'email viene trovata, viene generata un'eccezione con un messaggio di errore appropriato. Questo assicura che un ricercatore non possa essere inserito come valutatore se la sua email è già presente nella tabella `researchers`.
- Verifica se l'email inserita esiste già nella tabella `evaluators`, escludendo la riga corrente dall'operazione di controllo. Questo è necessario per consentire l'aggiornamento di altre colonne nella tabella `evaluators` senza generare un'eccezione di duplicazione email. Ad esempio, se un valutatore cambia il suo nome o cognome, l'aggiornamento dovrebbe essere consentito anche se l'email esiste già.
- Tuttavia, se l'email viene trovata su una riga diversa da quella corrente, viene generata un'eccezione per evitare la duplicazione dell'email tra i valutatori.
- Se nessuna corrispondenza viene trovata in entrambe le tabelle, l'email è considerata unica e il trigger restituisce la nuova riga per l'inserimento o l'aggiornamento.

2.4.3 Gestione dei ruoli e politiche di autorizzazione

Un altro step, già iniziato durante la raccolta ed analisi dei requisiti e che finalmente possiamo implementare è la gestione dei ruoli e delle politiche di autorizzazione. Come già accennato nel caso di EUResearchHub, sono stati definiti tre ruoli: **Admin**, **Researcher** e **Evaluator**.

Nello specifico i tre ruoli sono stati implementati come segue:

L'utente **Admin** ha privilegi di superutente, il che significa che può accedere a tutte le tabelle e eseguire qualsiasi operazione sullo schema EUResearchHub. Questo ruolo è utile per la gestione e la manutenzione del sistema e sarà limitato a un ristretto numero di utenti (ipoteticamente utenti incaricati e dipendenti dell'unione europea che si occupano di mantenere il servizio).

```

1 CREATE ROLE "Admin" SUPERUSER CREATEDB CREATEROLE NOINHERIT LOGIN NOREPLICATION
→ BYPASSRLS PASSWORD '****';
2 GRANT ALL PRIVILEGES ON SCHEMA "EUREsearchHub" TO "Admin";

```

Il ruolo del **Researcher** consente l'accesso alle tabelle relative ai documenti, ai tipi di documento, ai progetti, ai messaggi e alle associazioni tra ricercatori e progetti. I ricercatori possono inserire e modificare le informazioni relative ai loro progetti e documenti, ma non possono accedere alle tabelle degli evaluatori.

```

1 CREATE ROLE Researcher NOSUPERUSER NOCREATEDB NOCREATEROLE NOINHERIT LOGIN
→ NOREPLICATION NOBYPASSRLS PASSWORD '****';
2 GRANT SELECT ON TABLE "EUREsearchHub".document_types TO researcher;
3 GRANT INSERT, SELECT ON TABLE "EUREsearchHub".document_versions TO researcher;
4 GRANT INSERT, DELETE, SELECT ON TABLE "EUREsearchHub".documents TO researcher;
5 GRANT SELECT ON TABLE "EUREsearchHub".evaluation_reports TO researcher;
6 GRANT SELECT ON TABLE "EUREsearchHub".evaluation_windows TO researcher;
7 GRANT INSERT, SELECT ON TABLE "EUREsearchHub".messages TO researcher;
8 GRANT INSERT, DELETE, SELECT ON TABLE "EUREsearchHub".projects TO researcher;
9 GRANT INSERT, SELECT ON TABLE "EUREsearchHub".researchers_projects TO researcher;
10 GRANT SELECT ON TABLE "EUREsearchHub".researchers_messages TO researcher;

```

Il ruolo dell'**Evaluator** permette di accedere alle tabelle relative ai documenti, ai tipi di documento, alle valutazioni e alle finestre di valutazione. Gli evaluatori possono inserire report di valutazione, aggiungere commenti ai documenti inserendo una nuova versione del documento e aggiornare lo stato dei progetti, ma non possono modificare direttamente i documenti o i progetti stessi.

```

1 CREATE ROLE Evaluator NOSUPERUSER NOCREATEDB NOCREATEROLE NOINHERIT LOGIN
→ NOREPLICATION NOBYPASSRLS PASSWORD '4321';
2 GRANT SELECT ON TABLE "EUREsearchHub".document_types TO evaluator;
3 GRANT SELECT, INSERT ON TABLE "EUREsearchHub".document_versions TO evaluator;
4 GRANT SELECT ON TABLE "EUREsearchHub".documents TO evaluator;
5 GRANT INSERT, SELECT ON TABLE "EUREsearchHub".evaluation_reports TO evaluator;
6 GRANT SELECT ON TABLE "EUREsearchHub".evaluation_windows TO evaluator;
7 GRANT INSERT ON TABLE "EUREsearchHub".messages TO evaluator;
8 GRANT UPDATE, SELECT ON TABLE "EUREsearchHub".projects TO evaluator;
9 GRANT SELECT ON TABLE "EUREsearchHub".researchers TO evaluator;
10 GRANT SELECT ON TABLE "EUREsearchHub".researchers_messages TO evaluator;
11 GRANT SELECT ON TABLE "EUREsearchHub".researchers_projects TO evaluator;

```

Per quanto utilizzare ruoli permetta di evitare inconsistenze, ci sono anche alcune problematiche che possono sorgere se analizzato attentamente il problema. Le politiche di autorizzazione, consentono di imporre ulteriori ma necessarie restrizioni sull'accesso ai dati, e risolvere i problemi accennati. In particolare attraverso l'uso di **Row Level Security (RLS)**. RLS si può limitare l'accesso alle righe di una tabella in base a una condizione specificata nella politica.

Nel nostro caso, abbiamo forzato la Row Level Security sulla tabella `projects` e creato la politica `researcher_project_access_policy`. Questa politica limita l'accesso alle righe della tabella `projects` ai soli ricercatori associati a un determinato progetto. In questo modo, i ricercatori possono vedere e modificare solo i progetti a cui sono associati, garantendo una maggiore sicurezza e protezione dei dati.

```

1 ALTER TABLE "EUREsearchHub".projects FORCE ROW LEVEL SECURITY;
2 CREATE POLICY researcher_project_access_policy
3   ON "EUREsearchHub".projects
4     USING (id IN (SELECT fk_projects FROM "EUREsearchHub".researchers_projects WHERE
→   fk_researchers = current_setting('eu_research_hub.current_user_id')::integer));
5 ALTER TABLE "EUREsearchHub".projects FORCE ROW LEVEL SECURITY;

```

2.4.4 Performance: indici e materialized view

Anche la performance del database è un aspetto cruciale durante la progettazione fisica, poiché può influenzare direttamente l'esperienza utente e le prestazioni generali del sistema. Per ottimizzare le prestazioni del database, si possono utilizzare indici, materialized view e procedure. Di seguito viene analizzata l'implementazione degli indici e delle materialized view individuate ed implementate in EUResearchHub soffermandoci non solo sui vantaggi ma anche sui svantaggi che consapevolmente, abbiamo tenuto in considerazione. Questa scelta risulta in realtà momentanea; solo a webapp ultimata, si potrà effettivamente implementare la soluzione ideale definitiva tramite uno studio approfondito sulla distribuzione dei dati e log delle query (`ANALYZE` e `EXPLAIN`). Ad ogni modo si suppone che utilizzando le seguenti implementazioni si abbia un aumento della performance:

- `projects_status_idx`: questo indice viene creato sulla colonna `status` della tabella `projects`. Gli indici su colonne frequentemente utilizzate nelle clausole `WHERE` delle query possono velocizzare notevolmente le operazioni di ricerca. I vantaggi di questo indice includono un accesso più rapido ai dati per le query che filtrano i progetti in base allo stato (utile per i valutatori) e un miglioramento delle prestazioni delle query complesse che coinvolgono la tabella `projects`. Gli svantaggi includono un maggiore utilizzo dello spazio su disco e un potenziale rallentamento delle operazioni di inserimento, aggiornamento e cancellazione, poiché l'indice deve essere mantenuto coerente con i dati nella tabella. Ipottizzando però che queste tipologie di operazioni avvengono raramente o in momenti specifici (i.e. inizio/fine di una finestra di valutazione) si è deciso ad ogni modo di implementare questo indice per ottimizzare l'user experience nel resto del tempo.

```
1 CREATE INDEX projects_status_idx ON "EUResearchHub".projects (status);
```

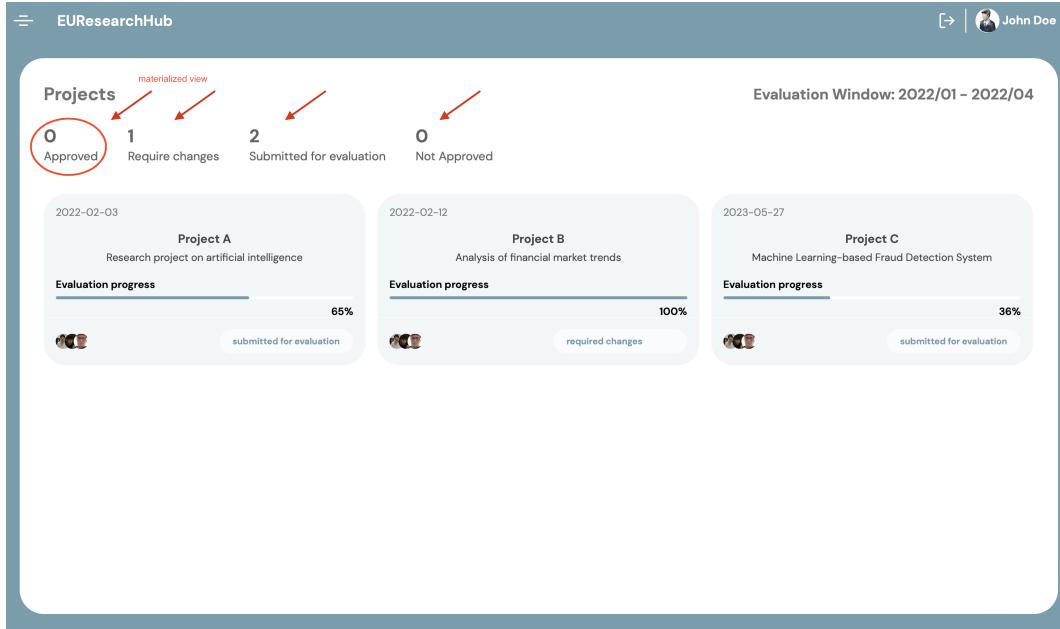
- `messages_fk_projects_idx`: questo indice viene creato sulla colonna `fk_projects` della tabella `messages`. Similmente all'indice precedente, l'obiettivo è di velocizzare le query che filtrano i messaggi in base al progetto associato. I vantaggi e gli svantaggi così come le scelte implementative sono simili a quelli dell'indice `projects_status_idx`. Attraverso questa scelta progettuale vogliamo privilegiare la visualizzazione dei messaggi più che l'inserimento essendo comunque non una normale chat sottoposta continuamente a inserimento di messaggi (i.e. WhatsApp) ma utilizzata più raramente.

```
1 CREATE INDEX messages_fk_projects_idx ON "EUResearchHub".messages (fk_projects);
```

- `projects_status_count`: questa materialized view memorizza il conteggio dei progetti raggruppati per stato individuati utili nella pagina relativi a tutti i progetti.

```
1 CREATE MATERIALIZED VIEW projects_status_count AS
2 SELECT status, COUNT(*) as count
3 FROM "EUResearchHub".projects
4 GROUP BY status;
```

Per chiarezza si faccia riferimento alla seguente immagine:



I vantaggi di questa includono un accesso rapido al conteggio dei progetti per stato e un miglioramento delle prestazioni per le query che richiedono queste informazioni come nel caso sopra illustrato. Gli svantaggi includono un maggiore utilizzo dello spazio su disco e la necessità di aggiornare la materialized view ogni volta che i dati nella tabella `projects` cambiano: indicamente alla fine di ogni finestra di valutazione. Per fare fronte a questo ultimo problema abbiamo individuato due soluzioni: aggiornare la materialized view al termine di un evaluation window o a seguito di ogni aggiornamento che la potrebbe influenzare. Abbiamo deciso di optare per la seconda opzione e quindi abbiamo inserito un nuovo trigger `refresh_project_status_count`, attivato solo a seguito di aggiornamento o inserimento di un nuovo status per ogni statement.

```

1 CREATE FUNCTION "EUResearchHub".refresh_project_status_count()
2 RETURNS trigger
3 LANGUAGE plpgsql
4 AS $function$
5 BEGIN
6   REFRESH MATERIALIZED VIEW projects_status_count;
7   RETURN NEW;
8 END;
$function$
10 CREATE TRIGGER refresh_project_status_count_insert
11 AFTER INSERT OR UPDATE
12 ON "EUResearchHub".projects
13 FOR EACH STATEMENT
14 EXECUTE FUNCTION refresh_project_status_count()

```

Capitolo 3

Sviluppo back-end



3.1 Scelta di Flask come framework

Nella fase di progettazione del back-end di EUResearchHu, abbiamo preso in considerazione diversi framework per lo sviluppo dell'applicazione web. Dopo un'attenta valutazione, abbiamo scelto Flask come framework principale per diverse ragioni. Flask è un micro-framework leggero e flessibile scritto in Python che fornisce un set di strumenti essenziali per lo sviluppo di applicazioni web. Inoltre, Flask offre una vasta gamma di estensioni che consentono di estendere le funzionalità di base del framework per adattarsi alle esigenze specifiche del nostro progetto. Ad esempio, abbiamo utilizzato Flask-WTF per gestire i moduli e i token CSRF, semplificando la gestione della sicurezza dei form e delle richieste.

3.2 Architettura dell'applicazione

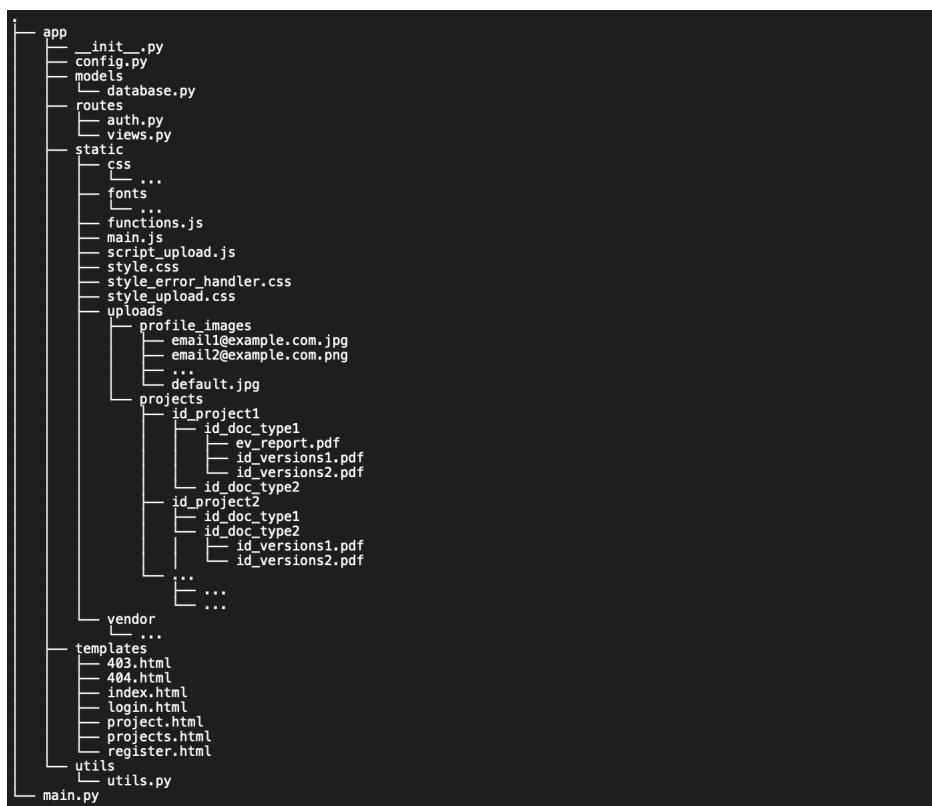
Per comprendere appieno il ruolo del back-end nel contesto dell'applicazione complessiva, è fondamentale analizzare l'architettura dell'applicazione nel suo insieme. L'applicazione segue un'architettura a tre livelli, composta da front-end, back-end e database.

3.2.1 Ruolo del back-end nell'architettura complessiva

Il back-end svolge un ruolo centrale nel coordinare le richieste provenienti dal front-end e l'interazione con il database. Nel nostro caso riceve le richieste HTTP provenienti dal front-end e le elabora in modo appropriato. Ciò include l'elaborazione dei parametri delle richieste, la validazione dei dati in ingresso e il reindirizzamento delle richieste verso le funzioni di gestione corrispondenti.

3.2.2 Organizzazione delle directory e dei file del progetto

Il sistema è organizzato in una struttura di directory ben definita, che favorisce una suddivisione logica dei file e delle risorse. Di seguito è riportato il dettaglio dell'albero delle directory principali:



3.2.3 Principali componenti del back-end

Come possiamo notare back-end del nostro progetto è costituito da diversi moduli e file di configurazione che lavorano in sinergia per fornire le funzionalità richieste. Ecco una panoramica delle principali componenti del back-end elencante della figura soprastante:

- `init.py`: Questo file è il punto di ingresso dell'applicazione Flask. Viene utilizzato per inizializzare l'applicazione e configurarla prima dell'avvio del server.
- `config.py`: Questo file contiene la configurazione dell'applicazione, come la chiave segreta per la gestione delle sessioni, le impostazioni del database e altre variabili di configurazione.
- `models/database.py`: Questo modulo definisce le classi di modello che rappresentano le tabelle del database. Utilizziamo SQLAlchemy per semplificare l'interazione con il database PostgreSQL.
- `routes/auth.py`: Questo modulo contiene le route e le funzioni associate per la gestione dell'autenticazione e dell'autorizzazione degli utenti, come il login, la registrazione e il logout.
- `routes/views.py`: Questo modulo contiene le route e le funzioni associate per la gestione delle pagine e delle funzionalità principali dell'applicazione, come la visualizzazione dei progetti e l'interazione con i documenti.
- `static/`: Questa directory contiene i file statici dell'applicazione, come fogli di stile CSS, script JavaScript e risorse come immagini e font.
- `templates/`: Questa directory contiene i template HTML utilizzati per generare le pagine dinamiche dell'applicazione.
- `utils/utils.py`: Questo modulo contiene funzioni di utilità che vengono utilizzate in diverse parti del back-end, ad esempio per la gestione delle operazioni di upload e manipolazione dei documenti.
- `main.py`: Questo file è il punto di avvio dell'applicazione. Viene utilizzato per eseguire il server Flask e rendere l'applicazione disponibile per le richieste in entrata.

In particolare, facciamo notare la cartella `uploads` che gioca un ruolo fondamentale nella gestione dei documenti e delle versioni. All'interno di questa cartella, sono presenti due sotto-cartelle principali:

- `uploads/profile_images`: Questa cartella contiene le immagini del profilo degli utenti registrati. Ogni immagine è salvata con il nome dell'utente corrispondente, consentendo un accesso rapido e diretto.
- `uploads/projects`: Questa cartella contiene i documenti associati ai progetti. È organizzata in base all'ID del progetto e contiene sotto-cartelle per ogni tipo di documento. All'interno di ciascuna cartella di tipo documento, sono presenti le diverse versioni del documento corrispondente.

La gestione dei documenti PDF avviene attraverso operazioni di upload, salvataggio e recupero dei file dal sistema di file. Le versioni dei documenti sono gestite creando una nuova cartella per ogni versione all'interno della corrispondente cartella di tipo documento. In questo modo, possiamo conservare una cronologia delle diverse versioni del documento e consentire agli utenti di accedere alla versione desiderata.

3.3 Interazione con il DBMS PostgreSQL

Per interagire con il database PostgreSQL, come già accennato, l'applicazione utilizza la libreria SQLAlchemy, che fornisce un'astrazione del livello di database e semplifica le operazioni di connessione, query e gestione dei dati. Nel file di configurazione dell'applicazione (`config.py`), è specificata l'URL del database PostgreSQL tramite la variabile `SQLALCHEMY_DATABASE_URI`. Questo URL contiene le informazioni necessarie per la connessione al database, come l'host, la porta, il nome del database, l'utente e la password. L'applicazione utilizza la configurazione del database per stabilire una connessione al database PostgreSQL utilizzando SQLAlchemy. Durante l'inizializzazione dell'applicazione, viene chiamato il metodo `db.init_app(app)` per collegare l'istanza del database SQLAlchemy all'applicazione Flask.

3.3.1 Utilizzo di SQLAlchemy per l'interazione con il database

Abbiamo optato per l'utilizzo di SQLAlchemy poiché così facendo l'interazione con il database PostgreSQL risulta fortemente semplificata. Il file `database.py` definisce modelli di dati Python utilizzando classi che estendono la classe `db.Model` di SQLAlchemy. Ogni classe di modello rappresenta una tabella nel database. Ad esempio, la classe `Projects` rappresenta la tabella `projects` nel database e contiene attributi che corrispondono alle colonne della tabella. L'ORM di SQLAlchemy gestisce automaticamente la mappatura tra gli oggetti delle classi di modello e le righe della tabella del database. Oltre a ciò SQLAlchemy risulta utile per EUResearchHub poiché permette di eseguire operazioni CRUD (Create, Read, Update, Delete) sul database utilizzando metodi e query forniti dalla libreria. Ad esempio, è possibile utilizzare il metodo:

```
1 db.session.query(Projects).filter_by(id=project_id).first()
```

per ottenere un oggetto `Projects` corrispondente a un determinato ID del progetto.

3.3.2 Esecuzione delle query principali e gestione dei risultati

In questa sezione, vengono illustrate alcune delle principali query eseguite per ottenere i dati necessari per il sistema.

1. Query per ottenere i progetti da visualizzare nella pagina `projects`:

```
1 if user_type == 'evaluator':
2     projects2show = Projects.query.all()
3     project_counts = ProjectsStatusCount.query.all()
4     for row in project_counts:
5         counts_by_status[row.status.value] = row.count
6 elif user_type == 'researcher':
7     projects2show = db.session.query(Projects).join(Researchers_Projects).filter(
8         (Researchers_Projects.fk_researchers == user.id).all()
9     )
10    project_counts = db.session.query(Projects.status,
11        func.count()).join(Researchers_Projects).filter(Researchers_Projects.
12            fk_researchers == user.id).group_by(Projects.status).all()
13    aux = {status.value: count for status, count in project_counts}
14    counts_by_status = {**counts_by_status, **aux}
```

In base al tipo di utente (evaluator o researcher), vengono eseguite query differenti per ottenere i progetti da visualizzare. Per gli evaluator, vengono ottenuti tutti i progetti e viene eseguita una query per contare il numero di progetti per ogni stato data dalla materialized views `ProjectsStatusCount`. Per i researcher, vengono ottenuti solo i progetti associati a quel researcher e viene eseguita una query per contare il numero di progetti per ogni stato specifico del researcher.

2. Query per ottenere le immagini del profilo dei ricercatori associati a ogni progetto:

```
1 researcher_profile_pictures = []
2 for project in projects2show:
3     project_id = project.id
4     profile_pictures = db.session.query(Researchers.profile_picture) \
5         .join(Researchers_Projects) \
6         .filter(Researchers_Projects.fk_projects == project_id) \
7         .all()
8     researcher_profile_pictures.append(profile_pictures)
9 Questa query viene eseguita per ogni progetto presente nella lista
  ↳ \imtaInlinetext{latex}{projects2show}. Per ciascun progetto, viene ottenuta
  ↳ la lista delle immagini del profilo dei ricercatori associati a quel
  ↳ progetto. Le immagini vengono quindi aggiunte a una lista per essere
  ↳ utilizzate nella generazione della pagina \imtaInlinetext{latex}{projects}.
```

3. Query per ottenere la percentuale di documenti valutati per ogni progetto:

```

1 evaluation_percentages = {}
2 for project in projects2show:
3     project_id = project.id
4     total_documents = Documents.query.filter_by(fk_project=project_id).count()
5     evaluated_documents = Evaluation_Reports.query.join(Documents).filter(
6         Documents.fk_project == project_id).count()
7
8     if total_documents > 0:
9         evaluation_percentage = int((evaluated_documents / total_documents) *
10             100)
11    else:
12        evaluation_percentage = 100
13
14    evaluation_percentages[project_id] = evaluation_percentage

```

Questa query viene eseguita per ogni progetto presente nella lista `projects2show`. Per ciascun progetto, viene calcolata la percentuale di documenti valutati rispetto al numero totale di documenti associati a quel progetto. I risultati vengono memorizzati in un dizionario dove la chiave è l'ID del progetto e il valore è la percentuale di valutazione.

4. Query per ottenere i messaggi associati a un progetto specifico:

```

1 messages = Messages.query.filter_by(fk_projects=project_id).all()

```

Questa query viene eseguita nella pagina `project` per ottenere tutti i messaggi associati a un determinato progetto. I messaggi vengono quindi utilizzati per mostrare una cronologia delle conversazioni relative al progetto.

5. Query per ottenere i ricercatori e i valutatori associati a ciascun messaggio di un progetto:

```

1 for message in messages:
2     researcher =
3         → Researchers.query.join(Researchers_Messages).filter_by(fk_messages =
4             → message.id).first()
5     evaluator = Evaluators.query.join(Evaluators_Messages).filter_by(fk_messages
6         → = message.id).first()
7     researchers.append(researcher)
8     evaluators.append(evaluator)

```

Questa query viene eseguita nella pagina `project` per ottenere i ricercatori e i valutatori associati a ciascun messaggio di un progetto. La query viene eseguita per ogni messaggio e si basa sulle tabelle di associazione `Researchers_Messages` e `Evaluators_Messages` per ottenere i ricercatori e i valutatori corrispondenti.

6. Query per ottenere i tipi di documento associati a un progetto:

```

1 documents = Documents.query.filter_by(fk_project=project_id).all()
2 docs = []
3 for doc in documents:
4     doc_type = Document_Types.query.filter_by(id=doc.fk_document_type).first()
5     docs.append(doc_type)

```

Questa query viene eseguita nella pagina `project` per ottenere i tipi di documento associati a un progetto. La query recupera tutti i documenti associati al progetto e per ciascun documento ottiene il tipo di documento corrispondente dalla tabella `Document_Types`.

7. Query per ottenere le versioni di un documento selezionato:

```

1 versions = Document_Versions.query.filter_by(fk_document=document.id).all()

```

Questa query viene eseguita nella pagina `project` quando viene selezionato un documento specifico. La query recupera tutte le versioni associate al documento selezionato dalla tabella `Document_Versions`.

3.4 Sicurezza del back-end

3.4.1 Utilizzo di Flask-WTF per gestire i moduli e i token CSRF

Nel sistema, viene utilizzato il modulo Flask-WTF per gestire i moduli e i token CSRF (Cross-Site Request Forgery). Flask-WTF semplifica la gestione dei moduli HTML fornendo funzionalità di validazione e protezione contro attacchi CSRF. La protezione CSRF è implementata utilizzando l'estensione `CSRFProtect` fornita da Flask-WTF.

3.4.2 Meccanismi di autenticazione e autorizzazione

Per garantire sicurezza, vengono implementati meccanismi di autenticazione e autorizzazione per consentire l'accesso agli utenti registrati. La route `/login` gestisce il processo di accesso. Quando un utente invia un modulo di accesso con un'email e una password valide, il codice effettua una verifica delle credenziali e autentica l'utente se i dati sono corretti. Vengono effettuate query al database per controllare se l'email appartiene a un valutatore o a un ricercatore, quindi viene utilizzata la funzione `check_password_hash` per confrontare la password fornita con la password memorizzata nel database. Se l'autenticazione ha successo, l'utente viene loggato utilizzando `login_user(user)` e reindirizzato alla pagina dei progetti e utilizzando una nuova connessione alla base di dati secondo il tipo di utente..

3.4.3 Salting e hashing delle password degli utenti

La funzione `generate_password_hash` del modulo `werkzeug.security` viene utilizzata per generare un hash della password. L'hash viene quindi memorizzato nel campo `password` del modello di dati per il valutatore o il ricercatore. Durante il processo di accesso, la funzione `check_password_hash` viene utilizzata per confrontare la password fornita dall'utente con l'hash memorizzato nel database. Questo approccio aumenta la sicurezza delle password degli utenti evitando di memorizzarle in forma di testo semplice.

3.4.4 Protezione da SQL Injection

Per prevenire attacchi di SQL Injection, viene utilizzato l'Object Relational Mapping (ORM) fornito da SQLAlchemy per eseguire query parametrizzate al database.

Capitolo 4

Implementazione front-end



Il frontend gioca un ruolo fondamentale per EUResearchHub. In questo capitolo, esploreremo l'importanza dell'interfaccia utente e dell'esperienza utente all'interno del contesto della nostra applicazione. L'interfaccia utente e l'esperienza utente svolgono un ruolo cruciale nell'efficacia e nella facilità d'uso della web application. La corretta progettazione del frontend garantisce che i ricercatori e i valutatori possano interagire in modo intuitivo e efficiente con il sistema, facilitando la gestione dei progetti di ricerca e migliorando l'efficacia complessiva del processo di valutazione.

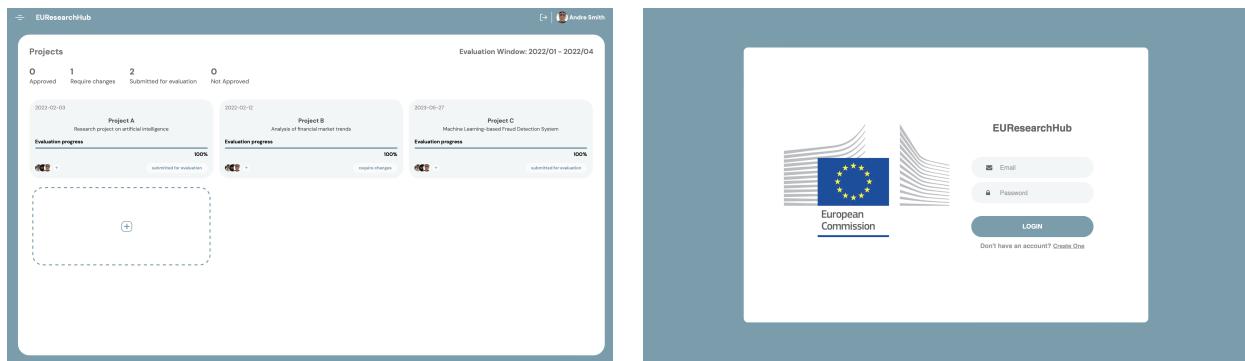
I progetto EUResearchHub è stato sviluppato utilizzando una serie di tecnologie front-end per garantire un'esperienza utente fluida e interattiva. Nel processo di sviluppo, sono state impiegate le seguenti tecnologie: HTML, CSS e JavaScript.

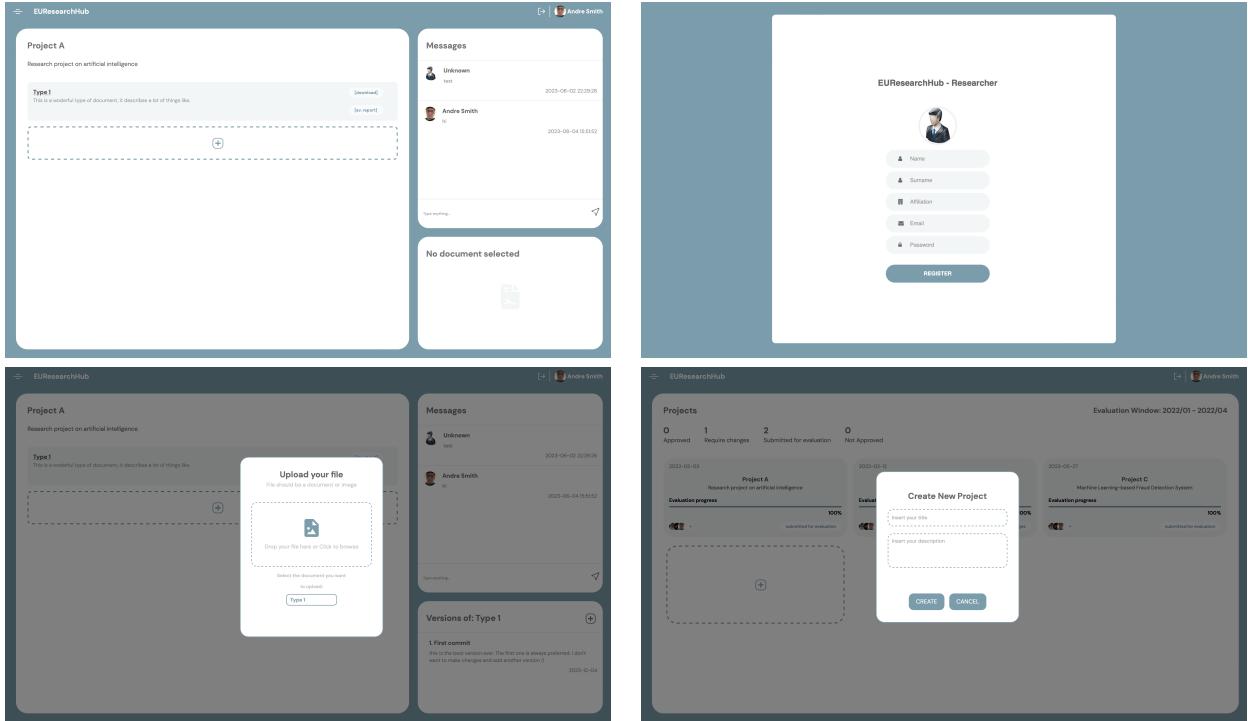
- **HTML (HyperText Markup Language)** è stato utilizzato per definire la struttura e la disposizione degli elementi sulla pagina web. Questo linguaggio di markup ci ha permesso di organizzare il contenuto in modo logico e gerarchico, rendendo più facile la creazione di una struttura coerente e accessibile.
- **CSS (Cascading Style Sheets)** è stato utilizzato per definire lo stile e l'aspetto visivo del sito web. Abbiamo utilizzato fogli di stile CSS per personalizzare il design, definendo colori, tipografia, layout e animazioni. In particolare, nel file HTML fornito sono state definite alcune regole CSS per gestire l'animazione di un'icona e il posizionamento degli elementi.
- **JavaScript** è stato utilizzato per aggiungere interattività e funzionalità dinamiche al sito web. Questo linguaggio di scripting ci ha permesso di gestire eventi, manipolare il contenuto della pagina in tempo reale e comunicare con il server per ottenere o inviare dati. Nel file HTML fornito, JavaScript è utilizzato per gestire il click su alcuni pulsanti, come il pulsante di logout e il pulsante di invio del messaggio.

Oltre alle tecnologie front-end di base, sono stati utilizzati anche alcuni strumenti aggiuntivi e framework per semplificare lo sviluppo. Tra questi:

- **Bootstrap**: un framework CSS che fornisce una serie di componenti predefiniti e stili predefiniti, rendendoci più rapido e semplice il processo di creazione di un'interfaccia responsiva e ben progettata.
- **jQuery**: una libreria JavaScript leggera e veloce che semplifica l'interazione con il DOM (Document Object Model) e fornisce una vasta gamma di funzionalità per la gestione degli eventi, l'animazione e le richieste AJAX.
- **Vue.js**: un framework JavaScript progressivo per la creazione di interfacce utente dinamiche. Vue.js ci ha permesso di facilitare la gestione dello stato dell'applicazione e la creazione di componenti riutilizzabili, migliorando l'organizzazione e la manutenibilità del codice.

4.1 Design finale





4.2 Integrazione tra front-end e back-end

L'integrazione tra il frontend e il backend nel progetto EUResearchHub avviene attraverso richieste HTTP effettuate dal client al server. Il frontend comunica con il backend per inviare e ricevere dati, consentendo all'utente di interagire con l'applicazione e visualizzare i risultati delle operazioni. Le richieste HTTP vengono gestite attraverso un'architettura API (Application Programming Interface), che definisce il modo in cui il frontend e il backend comunicano tra loro. Nel caso di EUResearchHub, vengono utilizzate diverse API per consentire all'utente di eseguire azioni come l'autenticazione, la ricerca di progetti di ricerca e l'invio di messaggi.

Durante queste interazioni, possono verificarsi situazioni di errore. Per gestire gli errori, il backend può generare messaggi di errore che vengono restituiti al frontend. Ad esempio, se si verifica un'eccezione durante l'esecuzione di un trigger o se il token di login o CSRF (Cross-Site Request Forgery) non è valido, il backend può generare un messaggio di errore e restituirlo al frontend.

Si inseriscono ora due esempi di richieste AJAX:

```

1  $.ajax({
2      url: '/api/get_doc_types',
3      type: 'POST',
4      datatype: 'json',
5      beforeSend: function(xhr) {
6          xhr.setRequestHeader('X-CSRF-Token',
7              $('meta[name="csrf-token"]').attr('content'))
8      },
9      success: (data) => {
10          $('#selectDocType').empty() data.forEach(item => {
11              $('#selectDocType').append(`<option class="inputdoctype"
12                  value="${item.id}">${item.name}</option>`)
13          })
14      }
15  })

```

In questo primo esempio una richiesta AJAX viene utilizzata per ottenere i tipi di documento da visualizzare in un'interfaccia modale. Quando l'utente fa clic su un pulsante per aprire la modale, viene scatenata una

funzione che effettua una richiesta AJAX al percorso `/api/get_doc_types` sul server. Questa richiesta viene inviata tramite il metodo POST e viene specificato che il tipo di dati atteso è JSON. Prima di inviare la richiesta, viene impostato l'header `X-CSRF-Token` con il valore del token CSRF presente nel meta tag della pagina. Nel backend, il server riceve la richiesta al percorso `/api/get_doc_types` e restituisce i tipi di documento come dati JSON. Nel successo della richiesta AJAX, i dati ricevuti vengono utilizzati per popolare una select box nel frontend con gli elementi appropriati.

```
1 var xhr = new XMLHttpRequest();
2 xhr.open("POST", "/add_participant", true);
3 xhr.setRequestHeader('Content-Type', 'application/json');
4 xhr.setRequestHeader("X-CSRFToken", csrfToken); // Include the CSRF token in the
   ↪ request headers
5 xhr.onreadystatechange = function() {
6     if (xhr.readyState === 4) {
7         console.log(xhr.responseText);
8         window.location.reload(); // Reload the page after the AJAX request completes
9     }
10 };
11 xhr.send(JSON.stringify({
12     projectId: projectId,
13     email: email
14 }));
15 }
```

In questo secondo esempio, invece, la funzione `addParticipant` viene utilizzata per inviare una richiesta AJAX per aggiungere un partecipante a un progetto. Viene creato un oggetto XMLHttpRequest e viene impostato il metodo, l'URL di destinazione e l'intestazione `Content-Type` per indicare che i dati inviati sono in formato JSON. Viene inoltre incluso il token CSRF nell'intestazione `'X-CSRFToken'`. La risposta alla richiesta viene gestita nella funzione `onreadystatechange`, che controlla lo stato della richiesta e, se è completata (`readyState === 4`), esegue un'azione (in questo caso, viene stampata la risposta sulla console e la pagina viene ricaricata con eventuali messaggi di errore (i.e. utente già nel progetto o mail non esistente)).

Capitolo 5

Contributo al progetto



5.1 Processo di sviluppo

 *Il lavoro di squadra divide i compiti e moltiplica il successo.* 

La divisione del lavoro all'interno del nostro gruppo di progetto per EUResearchHub è stata caratterizzata da un approccio collaborativo e flessibile. Sebbene tutti e tre i membri del gruppo abbiano acquisito competenze in diverse aree, tra cui backend, frontend e progettazione della base di dati, ci siamo impegnati a lavorare insieme su tutti gli aspetti del progetto.

Durante il processo di sviluppo, abbiamo adottato la piattaforma di collaborazione GitHub come strumento principale per il controllo delle versioni, la gestione del codice. e anche assegnare task e pianificare il lavoro. Attraverso GitHub, siamo stati in grado di lavorare contemporaneamente sulle diverse componenti del progetto, creare branch separati per le funzionalità in sviluppo e gestire i conflitti durante i merge.

Per mantenere un'adeguata comunicazione e allineamento, abbiamo stabilito due sprint settimanali durante i quali ci siamo riuniti in chiamate di allineamento. Durante queste chiamate, abbiamo discusso i progressi fatti, le sfide incontrate e le attività da svolgere nella settimana successiva. Questo ci ha permesso di monitorare l'avanzamento del lavoro, individuare eventuali ritardi o problemi e prendere decisioni condivise per affrontarli.

5.2 Analitiche Github

L'utilizzo di GitHub ci ha anche fornito analitiche e statistiche utili per valutare il contributo individuale e collettivo al progetto. Attraverso le funzionalità di tracciamento di GitHub, abbiamo potuto monitorare il numero di commit effettuati da ciascun membro, le modifiche apportate, le task e il tipo di task portate a termine, problemi sollevati e risolti, nonché le attività di code review e testing. In allegato alcune statistiche:

