Valutazione di progetti di ricerca

Alessandro Celadon 882778, Alessandro Giuliani 882483, Tomislav Dzepina 883801

December 10, 2023

Contents

1 Introduzione

1.1 Descrizione dell'Applicazione

L'applicazione è stata sviluppata come una web application con un'architettura ben definita, composta dallo *stack* **PostgreSQL**, **Flask**, **React**, **NodeJS**. Durante il processo di sviluppo, è stato creato un database locale PostgreSQL per ciascuno dei membri del team, utilizzando pgAdmin. Inoltre, è stato implementato un database condiviso ospitato su **Docker** per scopi di testing. Il modello del database è stato definito in Python attraverso **SQLAlchemy ORM**,

seguendo un approccio in cui ogni tabella del database corrisponde a una classe con vari metodi per interagire con i dati.

La tecnologia principale utilizzata per la realizzazione del backend è Flask, insieme a SQLAlchemy. La scelta è stata quella di implementare una **REST API**, consentendo un'interfaccia di comunicazione flessibile e separando completamente l'implementazione del client e del server. Inoltre, è stato definito un sistema di autorizzazioni basato su **JSON Web Tokens (JWT)** per garantire un accesso sicuro e controllato alle risorse dell'applicazione.

Per quanto riguarda il frontend, la scelta di un'architettura RESTful ha permesso l'utilizzo di JavaScript per sviluppare l'interfaccia utente. Questo approccio ha consentito al team di concentrarsi inizialmente sullo sviluppo del backend, posticipando l'implementazione del frontend. L'intenzione è quella di utilizzare strumenti potenti come React per la creazione dell'interfaccia utente una volta completato il backend.

In sintesi, la separazione tra backend e frontend ha facilitato lo sviluppo e la manutenzione, consentendo al team di concentrarsi su ciascuna componente in modo indipendente.

2 Architettura dell'Applicazione

Come detto in precedenza l'applicazione si basa su uno stack composto da PostgreSQL, Flask, React, Nodejs. Viene descritta in seguito una veloce panoramica sulla struttura dell'applicazione.

2.1 Database

2.1.1 Progettazione

Il database relazionale è stato progettato seguendo i principi dell'approccio ad oggetti e tradotto in un modello concettuale. Questo processo è stato guidato dalle tecniche illustrate nel libro di basi di dati e nel primo modulo del corso. Al fine di garantire che i vincoli imposti dai requisiti di progetto rispecchiassero fedelmente il modello concettuale, sono state adottate specifiche scelte implementative.

La progettazione è stata documentata attraverso un grafico che rappresenta la struttura relazionale del database, evidenziando le tabelle, le relazioni, e i vincoli implementati. Questo grafico fornisce una chiara visione di come i dati sono organizzati nel sistema.

2.1.2 Creazione e configurazione

La *creazione* del database è avvenuta attraverso l'utilizzo di PgAdmin, uno strumento di amministrazione per PostgreSQL. La struttura del database è stata *popolata* attraverso l'uso di Python e il framework SQLAlchemy ORM. Ogni tabella nel database relazionale è stata mappata a una classe Python, la

quale incorpora metodi specifici per operazioni come la creazione e l'eliminazione dei dati.

Per garantire un ulteriore strato di sicurezza sull'integrità dei dati, i **TRIGGER** sono stati implementati direttamente da PgAdmin attraverso l'esecuzione di **script SQL**.

La configurazione per collegare il database al codice python è contenuta nel file **config.py** in cui viene caricato il dotenv e viene configurata l'app di flask per connettere l'environment al nostro database.

```
8
9   app = Flask(__name__)
10
11  CORS(app)
12  load_dotenv()
13
14  app.config['SECRET_KEY'] = os.environ.get('DB_SECRET_KEY')
15  app.config['SQLALCHEMY_DATABASE_URI'] = os.environ.get('DB_URL')
16  app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
17
18  db = SQLAlchemy(app)
19  bcrypt = Bcrypt(app)
20
```

Figure 1: Configurazione Flask app in config.py

2.1.3 Testing

Per il testing dell'applicazione, è stato utilizzato un database condiviso, ospitato su un dispositivo locale tramite Docker. Questo approccio ha consentito di valutare le funzionalità del sistema in un ambiente controllato prima della distribuzione in un contesto più ampio.

2.2 Server Flask

Flask è stato utilizzato per implementare una REST API, consentendo una separazione chiara tra frontend e backend. Ciò offre un maggiore controllo e facilita lo sviluppo futuro. Maggiori informazioni sull'api sono nella sezione 5, una visualizzazione dei vari endpoint è illustrata a fondo documento.

2.3 Frontend

L'utilizzo di un'architettura RESTful permette l'implementazione del frontend con JavaScript. Questa separazione agevola lo sviluppo indipendente del backend e del frontend.

3 Funzionalità Principali

Le funzionalità principali dell'applicazione possono essere suddivise in categorie in base al tipo di utente coinvolto nell'operazione. Un meccanismo basato su JWT, implementato attraverso un decorator, specifica i permessi di una determinata route. Ci sono 4 classi di decorators: `token_required`, `researcher_required`, `evaluator_required`, `admin_required`.

Il decorator è un wrapper che ha la funzione di validare il JWT in base al campo 'role' conservato in esso e grazie alla chiave segreta immagazzinata nel dotenv del server. `token_required` è il grado di permesso più basso, può essere eseguito da tutti i tipi di utenti. `researcher_required` e `evaluator_required` richiedono specificamente di essere quel tipo di utente o superiore (`admin_required`). `admin_required`, infatti, può accedere a tutti gli endpoint con un ordine più basso e agli endpoint specificati dal decoratore `admin_required`.

Ulteriori informazioni sull'autenticazione sono specificate nella sezione 7.2.

- **Ricercatori:** I ricercatori possono creare un progetto, specificando un titolo, una descrizione e vari file che saranno i documenti del progetto. Durante la creazione, il ricercatore può assegnare un tipo ai file caricati, specificando il tipo di documento (ethics deliverable, data management plan, ecc.). Inoltre, possono rivedere i documenti appena caricati e rimuoverne alcuni prima di caricare il progetto. Inizialmente, il progetto viene caricato come bozza (draft) e può essere successivamente aggiornato o sottoposto a valutazione. I ricercatori possono anche rivedere i progetti da loro creati e consultare i report di valutazione. Per ogni progetto, è possibile visualizzare tutte le versioni e i documenti associati, anche alle versioni precedenti. I ricercatori possono ottenere le finestre di valutazione per comprendere esattamente quando il submit può essere effettuato. Un ricercatore può fare il submit solo se la finestra di valutazione non è ancora iniziata.

- **Valutatori:** I valutatori hanno modo di esaminare tutti i progetti sottoposti durante la prima finestra di valutazione disponibile. Sono abilitati a stilare un report per ciascun progetto, qualora non ne sia già stato predisposto. Durante la creazione del report, è loro consentito visualizzare e scaricare l'intera documentazione relativa al progetto sottoposto a valutazione.

  Per procedere con la valutazione, i valutatori devono aggiungere un documento contenente un report che illustri la loro prospettiva sul progetto. Successivamente, sono tenuti a assegnare un voto compreso tra 0 e 10 a dieci parametri distinti. Tali valutazioni vengono poi mediate, ottenendo una media che costituirà il voto del report.

  I valutatori possono eseguire la valutazione solo nel caso in cui la finestra di valutazione sia già in corso. Il verdetto riguardante lo stato specifico di un progetto sarà generato da uno scheduler al termine della finestra di valutazione, come dettagliato nella sezione 8.

- **Admin:** L'admin ha il compito di poter creare ed eliminare le finestre di valutazione. Inoltre, ha la possibilità di poter visionare i progetti contenuti in esse e relativi report.

- **Finestre di Valutazione:** La finestra di valutazione ha un ruolo cardine nel processo di sottomissione e valutazione di un progetto. Ci focalizzeremo principalmente su due tipi finestre la "current_window" e la

"next_window". Nella "current" possiamo trovare tutti i progetti sottoposti a valutazione, i valutatori possono creare report solo per i progetti nella current, mentre la next serve per far sottomettere i progetti di ricerca. Inoltre quando una finestra di valutazione e' conclusa(ovvero la data di fine diventa la data di oggi), verranno presi tutti i progetti contenuti in essa e verranno valutati da uno scheduler che ha il compito di valutarli(spiegato meglio nelle sezione 8 `scheduler.py`.)

- **Versioni:** Il sistema di controllo delle versioni rappresenta una delle funzionalità implementate, volta a tracciare le modifiche apportate a un progetto in ogni istante temporale, con particolare attenzione ai documenti. Ciascuna versione contiene un riferimento al progetto a cui è associata, contenente soltanto metadati di minore rilevanza. Al contrario, ogni versione include le informazioni cruciali da monitorare, tra cui i documenti del progetto e lo stato della versione.

  In particolare, per accedere ai documenti di un progetto, ci si rivolge all'ultima versione ad essa associata. Quando un progetto viene sottoposto (submit), si genera una nuova versione che incorpora i dati della versione precedente, ma modifica lo stato della versione a 'submitted', in questo specifico contesto.

4 Progettazione Concettuale e Logica della Basi di Dati

4.1 Progettazione Concettuale

La progettazione della base di dati è stata concepita considerando i molteplici vincoli imposti dalle relazioni intrinseche tra le tabelle. Ad esempio, solamente i ricercatori hanno il privilegio di creare progetti, i quali vengono inizializzati con lo stato 'draft'; analogamente, solo i valutatori possono generare un report di valutazione, e così via.

Nel contesto della progettazione concettuale, per riflettere tali vincoli, è stata implementata una classificazione di alcune tabelle mediante una gerarchia di ereditarietà, attraverso l'uso di sottoclassi. La classe 'Utente' è stata suddivisa in due sottoclassi, 'Valutatore' e 'Ricercatore', ciascuna delle quali interagisce in modi distinti con le altre tabelle. Inoltre, i progetti sono stati categorizzati in diverse sottoclassi, tra cui 'draft', 'requires changes', 'submitted', 'approved', e 'not approved'. Ad esempio, le classi 'approved' e 'not approved' sono inaccessibili per modifiche tramite un report e sono consultabili esclusivamente dai ricercatori che hanno originariamente creato il progetto.

4.2 Progettazione Logica

Nella fase di progettazione logica, si è adottato il partizionamento a tabella unica per modellare le sottoclassi. Tale scelta è stata dettata da certe con-

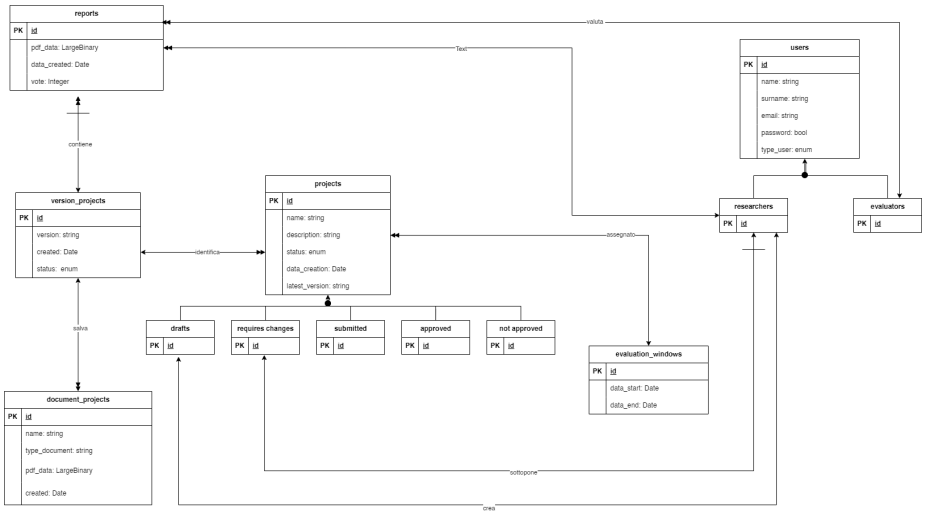


Figure 2: Progettazione Concettuale del Database

siderazioni pratiche, in particolare dalla mancanza di attributi distintivi per le sottoclassi. Ogni sottoclasse, infatti, non presentava attributi aggiuntivi rispetto a quelli della superclasse. Specificamente, nel contesto del partizionamento degli 'Utenti', la decisione di optare per una tabella unica è stata guidata dalla semplicità ottenuta nel recuperare le informazioni della superclasse, senza la necessità di transitare attraverso gli identificatori delle tabelle delle sottoclassi. Tuttavia, è importante sottolineare che il partizionamento a tabella unica ha comportato la rimozione di alcuni vincoli originariamente derivanti dalle relazioni. Nel corso dell'implementazione della REST API, ci si è impegnati a considerare attentamente tutti i vincoli che potevano andare persi durante questa trasposizione. Ci siamo assunti la responsabilità di reimplementarli attraverso la logica della API e alcuni triggers.

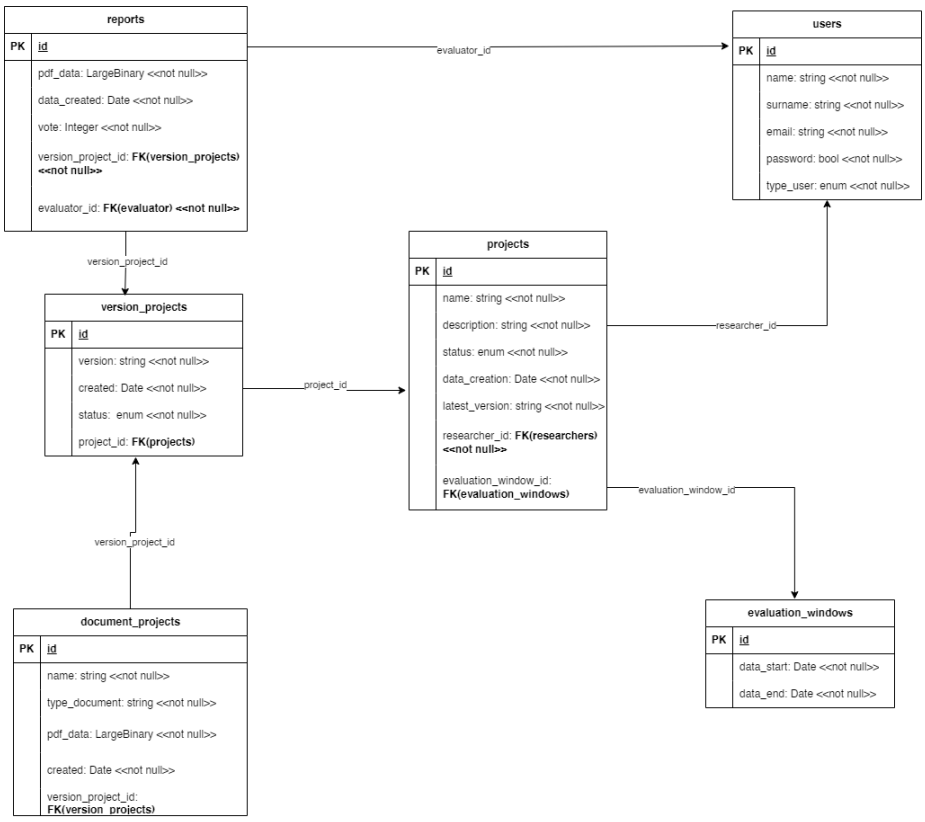


Figure 3: Progettazione Logica ad oggetti del Database

5 REST API

La logica del backend dell'applicazione è implementata mediante una REST API. Gli endpoint sono organizzati in vari moduli utilizzando Flask Blueprint.

La REST API offre un sistema di autenticazione basato su JWT, in cui i requisiti di accesso sono specificati attraverso dei decorators che considerano il ruolo dell'utente. La documentazione dettagliata della REST API è fornita attraverso le immagini illustrate a fondo documento.

```
app.register_blueprint(user_blueprint, url_prefix='/user')
app.register_blueprint(proj_blueprint, url_prefix='/projects')
app.register_blueprint(researcher_blueprint, url_prefix='/researchers')
app.register_blueprint(evaluators_blueprint, url_prefix='/evaluators')
app.register_blueprint(window_blueprint, url_prefix='/evaluation-window')
app.register_blueprint(project_version_blueprint, url_prefix='/version_project')
app.register_blueprint(unauth_user_blueprint, url_prefix='/unauth-user' )
app.register_blueprint(documents_blueprint, url_prefix='/documents')
```

Figure 4: (File app.py) Registrazione dei moduli della rest api nella flask app

6 Query Principali

Tutte le query dell'applicazione sono state implementate attraverso la sintassi SQLAlchemy ORM, la maggior parte di esse sono semplici query in cui viene utilizzato il filter_by, che abbiamo astratto in metodi delle classi dei Modelli, dove una query SQLAlchemy come Project.query.filter_by(id=id).first() viene astratta in un metodo statico Project.get_project_by_id(id) per nascondere eventuale implementazione a livello della REST API.

Una query più interessante a livello implementativo può essere quella effettuata nella **get_user_project_by_id(project_id)** situata in `models/project` che viene utilizzata per verificare che l'utente che richiama la query attraverso l'endpoint, non specifichi un progetto che non sia suo.

Altre due query molto usate sono la get_current_window() e la get_next_window() situate in `models/evaluation_window`. Esse hanno un ruolo specifico nel recupero delle informazioni della finestra per effettuare operazioni di inserimento o modifica dei progetti(come nella fase di submit di un progetto).

7 Principali Scelte Progettuali

7.1 Politiche di Integrità

Per garantire coerenza ed affidibilita' nei dati, abbiamo implementato una serie di poilitiche d'integrità sia a livello database che livello applicativo

7.1.1 Unique Constraint

- **Users**: per evitare di registrare uno user con la stessa email
- **Researcher**: per avere un solo researcher per user id
- **Evaluator**: per avere un solo evaluator per user id
- **Report**: per evitare che un evaluator crei più report per un singolo progetto

7.1.2 Check Constraint

- **EvaluationWindow**: per evitare che la data di inizio sia maggiore rispetto la data di fine
- **Report**: per evitare che il voto non sia compreso tra 0 e 10

7.1.3 Triggers

Malgrado l'accesso ai dati sia rigidamente controllato tramite l'esclusivo utilizzo della REST API, abbiamo ritenuto opportuno implementare dei trigger attraverso l'ambiente di gestione pgAdmin. Tale decisione è stata presa con l'obiettivo di garantire un maggiore controllo sull'integrità dei dati e di agevolare il processo di sviluppo.

- **Project**: before update, per evitare che un progetto venga modificato quando si trova in stato APPROVED o NOT_APPROVED
- **EvaluationWindow**: before insert, per controllare che la finestra che stiamo creando non sovrascriva altre finestre e che non venga creata nel passato.
- **Report**: before insert, per far si che il report venga creato solo per progetti submitted e nella finestra di valutazione corrente

7.2 Definizione di Ruoli, Autenticazione ed Autorizzazioni

L'autenticazione e l'autorizzazione sono gestite attraverso JSON Web Tokens, come si può vedere nell'endpoint di login, quando un utente l'effettua con successo, viene creato un payload contenente informazioni quali nome, cognome, ruolo, id dell'utente e data di scadenza del token. Questo payload viene quindi codificato in un JWT, con l'ausilio dell'algoritmo HS256 e la secret key, per poi essere restituito come risposta della richiesta ed eventualmente salvato ed inviato negli headers degli endpoint che richiedono accesso protetto. Lato implementativo , abbiamo creato un decorator(posizionato in utils/middleware) che verifichi l'autorizzazione in base ai tre principali ruoli ADMIN, EVALUATOR e RESEARCHER.

7.3 Uso di Indici

Abbiamo optato per non utilizzare indici nel nostro progetto per evitare un aumento dei costi nelle operazioni di gestione dati, come inserimento, cancellazione e aggiornamento. Questa scelta si basa sul fatto che la frequenza elevata di tali operazioni sulle tabelle interessate renderebbe inefficiente l'uso degli indici. Inoltre, le dimensioni ridotte delle tabelle e la limitata occupazione di memoria non giustificano l'introduzione degli indici per migliorare le prestazioni complessive del sistema.

7.4 Transazioni

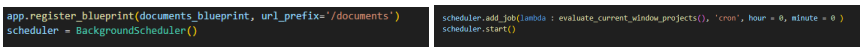
Nel nostro progetto, abbiamo scelto di utilizzare le transazioni per gestire tutte le operazioni sui dati. Per implementare ciò, sfruttiamo una funzionalità offerta dalla libreria SQLAlchemy. Questo ci consente di evitare uno stato inconsistente dei dati all'interno del database e di operare con dati aggiornati alle reali ultime modifiche effettuate su di essi.

Le operazioni vengono eseguite dai metodi da noi implementati nel file `utils/db_utils.py`. I commit sono contornati da un "try : except: " dove nel blocco "except", eseguiamo un rollback() nel caso in cui l'operazione di commit() non vada a buon fine. In questo modo, l'eventuale aggiunta, aggiornamento o eliminazione di un elemento della sessione viene annullata e ripristinata la situazione precedente. In alcuni metodi abbiamo eseguito un flush() della sessione, data la necessità di avere dati sincronizzati con il database senza però effetturare un commit(), cosi da wrappare l'operazione in una singola transazione.

8 Ulteriori Informazioni

Informazioni dettagliate sulle scelte tecnologiche specifiche, inclusi dettagli sulle librerie utilizzate e qualsiasi altra informazione rilevante per comprendere appieno il progetto.

- **scheduler.py**: Per valutare i progetti abbiamo utilizzato una libreria esterna **apscheduler**, che tramite il modulo BackGroundScheduler fornisce l'implementazione di uno scheduler che viene eseguito in background con funzionalità di cronjob. Ogni giorno alle 23:59 controlla se la finestra di valutazione corrente sta per terminare, in tal caso viene invocata una funzione che valuta i progetti come segue, tutti i progetti con i report li valuta in base alla valutazione democratica dei valutatori, per poi modificare lo stato del progetto appena valutato in base al voto >= 7 Approvato, <= 3, Non approvato e tra 3 e 7 Richiesta di Modifiche. I progetti senza report vengono messi automaticamente in Richiesta di Modifiche.



- **login**: La route "/login" è definita nel file`api/user_routes.py` e gestisce la validazione delle credenziali dell'utente. Durante questa operazione, vengono inviate al server un'email e una password associate a un utente specifico. Poiché la password è memorizzata in forma crittografata nel database e la verifica dell'utente avviene durante la fase di registrazione, si fa uso della libreria **bcrypt**. Questa libreria impiega un algoritmo di hash per decodificare e convalidare la password dell'utente in modo sicuro.

9 Contributo al Progetto (Appendice)

Ciascun membro del progetto ha lavorato sul codice e sulla progettazione. Una migliore panoramica del contributo al progetto si puo visualizzare sulla relativa repository di GitHub al link: https://github.com/tomislav98/DB-progetti-di-ricerca.git

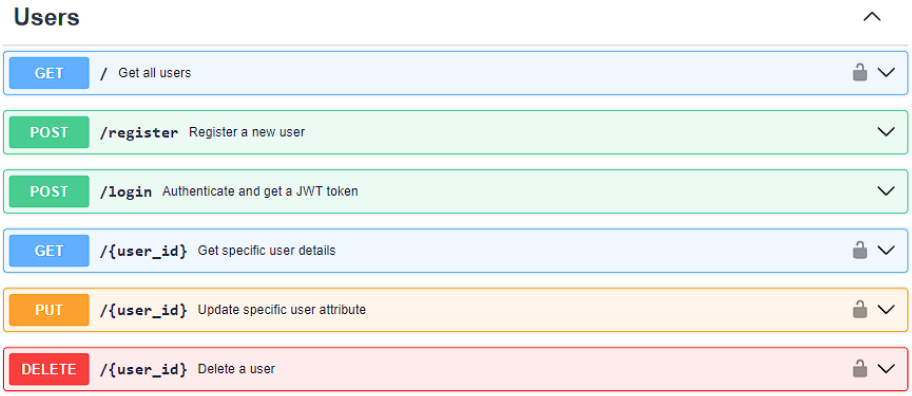


Figure 5: User routes

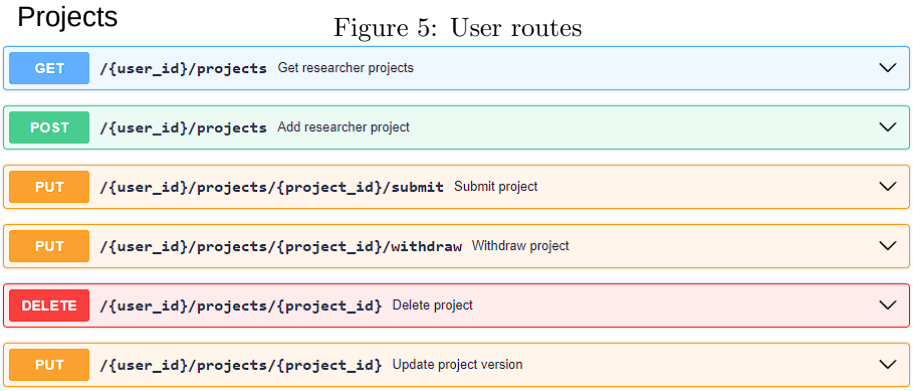


Figure 6: Project routes

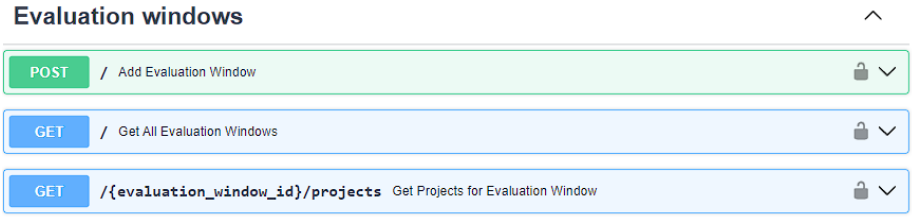


Figure 7: Windows routes

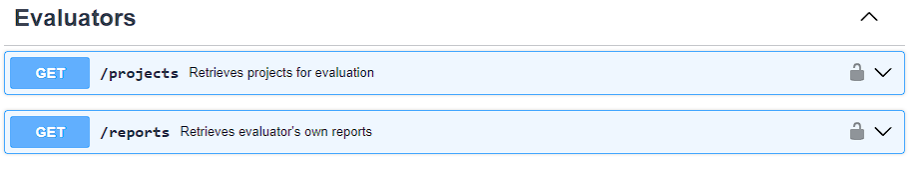


Figure 8: Evaluators routes

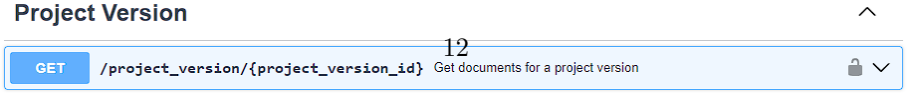


Figure 9: Evaluators routes