



Relazione di progetto

Gruppo DataBaseLegends

Giovanni Costa 880892 e Andrea Munarin 879607 | Basi di Dati mod2 | 29/08/2021

Indice:

1. [Introduzione](#)
2. [Funzionalità principali](#)
 - 2.1 [Init](#)
 - 2.2 [Form view](#)
 - 2.3 [Form edit](#)
 - 2.4 [Form add](#)
 - 2.5 [Users](#)
 - 2.6 [Error page](#)
3. [Progettazione base di dati](#)
 - 3.1 [Modello Concettuale](#)
 - 3.2 [Modello Logico](#)
4. [Query principali](#)
 - 4.1 [Possibili domande importabili](#)
 - 4.2 [Le query più usate](#)
 - 4.3 [Modifica delle domande di un form](#)
 - 4.4 [Visualizzazione delle risposte ad un form](#)
 - 4.5 [Query di cancellazione dei form](#)
5. [Principali scelte progettuali](#)
 - 5.1 [Trigger definiti](#)
 - 5.2 [Sicurezza: Ruoli e prevenzione da SQL injection](#)
 - 5.3 [Performance](#)
 - 5.4 [Utilizzo delle transazioni e ORM](#)
6. [Librerie necessarie per il funzionamento dell'app](#)

1. Introduzione

Il tema scelto per il progetto sullo sviluppo dell'applicazione web è stato il primo: "Creazione di questionari online".

La web application implementa la creazione di questionari nello stile di Google Forms. L'applicazione permette la creazione di domande, di tipo diverso (risposta singola, risposta multipla o risposta aperta), da combinare all'interno di un unico questionario da rendere accessibile ad una community di utenti. Ogni domanda possiede uno o più tag che riferiscono all'argomento che tratta.

Le domande, inoltre, possono essere rese obbligatorie per un determinato questionario ed in più vi è la possibilità di permettere il caricamento di file da parte dell'utente (solo su domande aperte).

Il creatore di un questionario ha anche accesso ad un'interfaccia di analisi dei dati per visionare le statistiche sulle risposte fornite dalla community e scaricare tali risposte in formato CSV.

Possono inoltre essere creati degli amministratori per applicazione che oltre alle normali funzioni che ha un utente, avranno delle interfacce di gestione ad-hoc.

Il backend dell'app è stato sviluppato in Python, con l'ausilio del popolare framework Flask e di alcune librerie, quali Flask-Security, per la gestione degli utenti e della sicurezza, e SQLAlchemy, per interfacciarsi col database sottostante. Queste verranno descritte successivamente.

Per l'organizzazione e lo sviluppo del progetto sono state anche ampiamente utilizzate le Blueprint.

Il frontend è stato sviluppato in HTML, JavaScript e CSS, con l'ausilio del framework Bootstrap e della libreria Chart.js, per i grafici.

2. Funzionalità principali

Vengono ora descritte le funzionalità principali dalla web application suddivise per categorie.

Ogni categoria cerca di riassumere dei file Python presenti all'interno del progetto:

- **Init** descrive l'inizializzazione della web app
- **Form view** descrive tutte le informazioni riguardanti la visualizzazione dei form e le risposte degli utenti
- **Form edit** comprende tutte le funzioni per la modifica specifica degli elementi del form
- **Form add** descrive l'aggiunta o la creazione di elementi
- **Users** illustra come vengono gestiti gli utenti

2.1 Init

L'inizializzazione dell'app si compone della configurazione di tutti gli elementi che verranno successivamente utilizzati:

- Connessione al database effettuata nel file "database.py", dopo aver opportunamente modificato delle variabili per il collegamento al suo interno (in questo caso specifico il BD utilizzato è PostgreSQL)
- Creazione delle tabelle nel database con i corrispettivi vincoli e dei trigger (scritti in PL/pgSQL) fatto dal sistema ORM di SQLAlchemy in "models.py"
- Inizializzazione di tutte le costanti di sistema per permettere un corretto funzionamento della libreria Flask-Security
 - Flask-security è una vasta libreria software, comprendente per esempio anche Flask-Login, che gestisce l'autenticazione e l'autorizzazione di utenti all'interno dell'app, fornendo funzionalità di sicurezza interessanti come la validazione di un utente tramite mail di conferma, gestione di token di accesso e criptatura della password nel database, e non solo

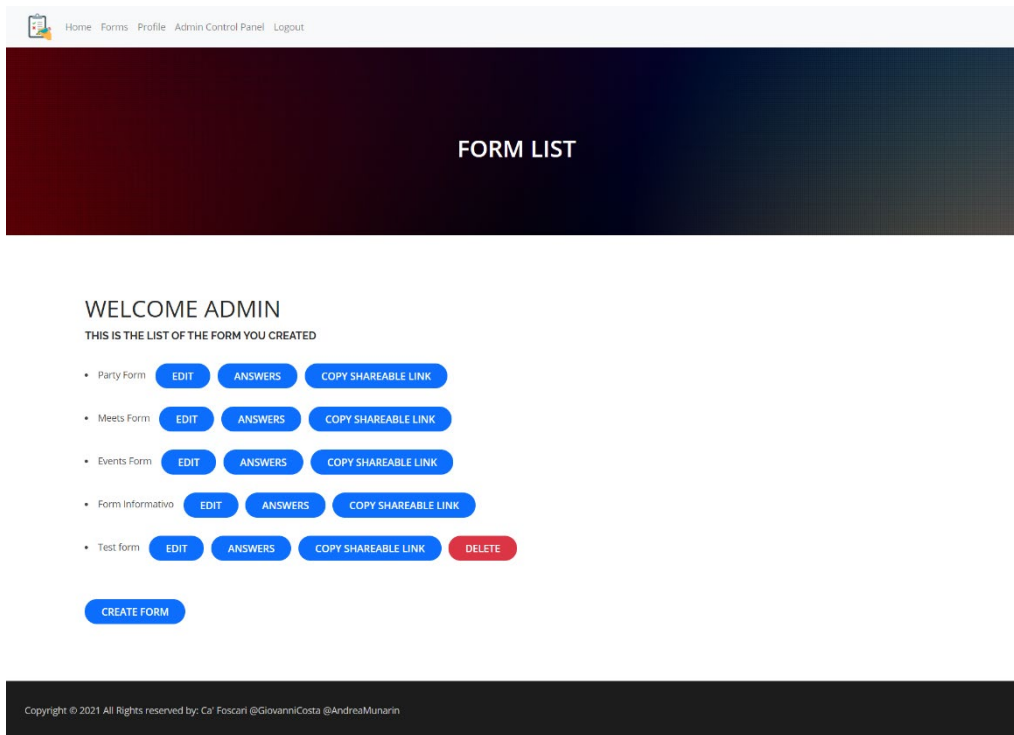
- Per le funzioni di autenticazione e sicurezza, quali criptatura e token access, è necessario impostare una secret key ed un “salt” (sequenza di caratteri casuali aggiuntiva).
Queste due stringhe vengono generate casualmente all’avvio e vengono memorizzate nel file “.env” (viene creato al momento) in modo da non essere visibili nel codice sorgente
- Creazione ed inserimento all’interno di database di alcuni dati che lo popolano inizialmente:
 - Ruoli “Admin”, “Standard User” e “SuperUser”, che verranno descritti nel dettaglio successivamente
 - Un utente SuperUser/admin e due standard user
 - 4 template di form (sono state inserite circa 25 domande ed alcuni tag possibili)
 - Alcune ipotetiche risposte ai vari form

2.2 Form view

Nella Blueprint “form_view_BP.py”, che ha come prefisso di default nell’url /form, si notano una serie di endpoint riguardanti la visualizzazione del form e delle risposte associate ad essi:

- /
Mostra l’elenco di tutti i form creati dall’utente insieme agli specifici pulsanti per:
 - entrare nella schermata modifica del form
 - cancellare il form
 - copiare nella clipboard il link condivisibile del form per poterlo inviare agli utenti della community in modo che possano compilarlo

Inoltre, è presente anche un pulsante per creare nuovi form, con la possibilità di selezionare l’importazione di un template di base o di un altro form creato precedentemente







route: /

- `/<form_id>/viewform`

Permette la compilazione del form per gli altri utenti (il creatore viene reindirizzato a edit form, visto che non avrebbe senso che egli rispondesse al proprio form)

Nel caso si acceda a questo endpoint in POST sono presenti varie utilities per validare input particolari, come checkbox vuote, mancato caricamento di file o rifiuto se essi presentano un'estensione non consentita

Se il creatore del form marca una domanda come obbligatoria all'interno della pagina html di visualizzazione lo specifico tag presenterà l'attributo html "required" in modo da non permettere all'utente di inviare risposte vuote in quei campi (Focus: se la domanda aperta marcata come obbligatoria ha anche la possibilità di aggiungere un file, anche esso dovrà essere necessariamente caricato)

 Home  Forms  Profile  Logout

FORMS LIST

WELCOME PIPPO

THIS IS THE FORM: **MEETS FORM**.

MEETING

HERE THE QUESTIONS:

Nome

Cognome

Età

Sesso

☐ M

☐ F

Mail

In quanti parteciperete?

Hai intolleranze alimentari, se sì quali? *

Carica il tuo CV

Select File:

Scegli file

Nessun file selezionato

Cancel File

Submit

Reset

Copyright © 2021 All Rights reserved by: Ca' Foscari @GiovanniCosta @AndreaMunarin

route: /<form_id>/viewform

- /<form_id>/answers e /<form_id>/download_csv
Mostra le varie risposte date dagli utenti ad uno specifico form
Come anticipato nell'introduzione, è possibile anche visionare dati sulle risposte e scaricare le stesse in un file con formato CSV:
 - Per quanto riguarda le domande a risposta chiusa singole viene visualizzato un grafico a torta con le percentuali di utenti che hanno dato una determinata risposta
 - Per le domande a risposta multipla è presente invece un istogramma

- Il file contenente l'insieme delle risposte è così strutturato → utente, domanda, risposta



route: /<form_id>/answers

- /answers/<answer_id>

Questo endpoint è utilizzato per visualizzare i file caricati dagli utenti che hanno risposto ad una specifica domanda nel form. Una volta aperto il tab è anche possibile scaricare il file mediante le funzioni messe a disposizione dal browser

2.3 Form edit

Nella Blueprint “form_edit_BP.py”, che ha come prefisso di default nell'url /form, si notano invece una serie di endpoint riguardanti la modifica di elementi presenti nel database:

Home Forms Profile Admin Control Panel Logout

EDIT FORM

WELCOME ADMIN

THIS IS THE FORM: **TEST FORM.**

MODULO UTILIZZATO A SCOPO DI TEST

[EDIT FORM INFO](#)

HERE THE QUESTIONS:

1. Nome [EDIT QUESTION](#) [DELETE](#)

☐ Allows file
☒ Mandatory

2. Cognome [EDIT QUESTION](#) [DELETE](#)

☐ Allows file
☒ Mandatory

3. Inserisci l'orario che preferisci [EDIT QUESTION](#) [DELETE](#)

☐ Allows file
☐ Mandatory

4. In quanti parteciperete? [EDIT QUESTION](#) [DELETE](#)

☐ Allows file
☐ Mandatory

5. Scegli i giorni della settimana che preferisci [EDIT QUESTION](#) [DELETE](#)

☐ Mandatory

- ☐ lunedì
- ☐ martedì
- ☐ mercoledì
- ☐ giovedì
- ☐ venerdì
- ☐ sabato
- ☐ domenica

6. Parteciperai all'evento? [EDIT QUESTION](#) [DELETE](#)

☐ Si
☐ No
☒ Mandatory

7. Carica una tua foto [EDIT QUESTION](#) [DELETE](#)

☒ Allows file
☒ Mandatory

[ADD QUESTION](#)

Copyright © 2021 All Rights reserved by: Ca' Foscari @GiovanniCosta @AndreaMunarin

pagina per la modifica del form

- /<form_id>/edit
Mediante questa interfaccia il creatore del form può visualizzarlo e modificarlo, sfruttando i vari pulsanti messi a disposizione:

- Edit form info: per modificare solamente nome e descrizione
 - Edit Questions
 - Delete Questions, che non elimina la domanda dal database, ma rimuove solamente il collegamento tra domanda e form (come descritto nel punto “Progettazione Base di Dati” il database è stato strutturato in modo che le domande vengano memorizzate solo una volta e che siano quindi “condivise” con riferimenti, quando però viene richiesta una modifica ne viene creata una nuova ed essa viene collegata allo specifico form opportunamente)
 - mandatory e allows_file, che permettono di rendere una domanda obbligatoria oppure di permettere il caricamento di un file relativo a quella domanda (quest’ultimo solo su domande aperte)
- /<form_id>/<question_id>/flag
 Tramite questo endpoint viene gestita l’attivazione o la disattivazione degli attributi mandatory e allows_file delle varie domande. Per permettere il funzionamento di queste operazioni è stato necessario implementare le relative checkbox nell’interfaccia html come form html indipendenti per ogni domanda. In particolare, è stato utilizzato questo escamotage: è stato inserito oltre al tag html input “checkbox” anche un input tag “hidden” per inviare all’endpoint tramite richiesta POST più dati sulla selezione o deselezion della checkbox
 - /<form_id>/editMainInfo
 Permette invece la modifica di nome e descrizione di un determinato form (non è possibile lasciare questi campi vuoti)
 - /<form_id>/<question_id>
 Garantisce la modifica una specifica domanda all’interno di un determinato form. Questa è una delle funzioni legate alle route più complesse e sfrutta inoltre una funzione esterna contenuta nel file “form_function.py”.

La pagina di visualizzazione che permette la modifica delle domande è molto simile a quella che permette l'aggiunta di nuove domande: si basano tutte su dropdown menù dinamici che appaiono in base a quale risposta è stata inserita nei precedenti (sfruttando JavaScript).

Inoltre, questa schermata di edit permette di modificare la domanda per interno o solamente le singole possibili risposte nel caso di una domanda a risposta chiusa.

Si descrive ora più nel dettaglio il funzionamento:

- Innanzitutto, le domande possono essere importate oppure no, e le domande importabili sono quelle già presenti nel database create dall'utente corrente o da un utente admin, oppure sono quelle di base create inizialmente nella funzione di inizializzazione
- Se una domanda non viene importata invece è necessario crearla da zero; occorre quindi scegliere un insieme di tag (o uno solo) e il tipo di domanda.

In base all'opzione selezionata si richiederà di inserire un nuovo tag nel caso sia stato selezionato "nuovo tag" nel dropdown menù e successivamente le possibili risposte nel caso si tratti di una domanda a risposta chiusa

La funzione in POST di memorizzazione, controllando tutti i vari casi, salverà/aggiognerà nel database, attraverso le varie transazioni, i diversi dati inseriti dall'utente

The current question is:
Nome

WOULD YOU LIKE TO IMPORT SOME EXISTING QUESTIONS?

No ▾

THIS QUESTION WOULD BE MANDATORY?

☒

SELECT THE QUESTION TAG ARGUMENT (OR SELECT MULTIPLE WITH CTRL/CMD)

Organizzazione
Altro
Scuola
Lavoro
Animali

SELECT A QUESTION TYPE

Open ▾

INSERT THE QUESTION TEXT

Dove lavori?

WOULD YOU LIKE TO ALLOW THE POSSIBILITY TO ADD FILES FOR THIS QUESTION?

No ▾

Submit

route: /<form_id>/<question_id>

2.4 Form add

Nella Blueprint “form_add_BP.py” , che ha come prefisso di default nell'url /form, si trovano poi una serie di endpoint riguardanti la creazione di nuovi elementi:

- /form_create

Consente di creare un nuovo form da zero, importandolo da uno dei 4 form template iniziali oppure da uno dei form dell'utente precedentemente creati, permettendo di specificare nome e descrizione diversi

WOULD YOU LIKE TO IMPORT A FORM TEMPLATE?

Yes ▾

Form Name

Nome

Descrizione Form

Descrizione

Template Name

Party Form ▾

Template Questions Preview:

1. Nome
2. Cognome
3. Inserisci l'orario che preferisci
4. In quanti parteciperete?
5. Scegli i giorni della settimana che preferisci
 - Lunedì
 - martedì
 - mercoledì
 - giovedì
 - venerdì
 - sabato
 - domenica
6. Parteciperai all'evento?
 - Sì
 - No

Submit

- /<form_id>/add_question

Funziona esattamente come edit_question solamente che non permette (ovviamente) la creazione di sole singole risposte. Sfrutta infatti lo stesso template html e le stesse funzioni presenti in form_function.py

The current question is:

Nome

WOULD YOU LIKE TO IMPORT SOME EXISTING QUESTIONS?

Yes ▾

THIS QUESTION WOULD BE MANDATORY?



SELECT THE QUESTION TAG ARGUMENT

Altro ▾

SELECT THE QUESTION

Valuta questo sondaggio ▾

Question Preview:

Valuta questo sondaggio

- 1
- 2
- 3
- 4
- 5

Submit

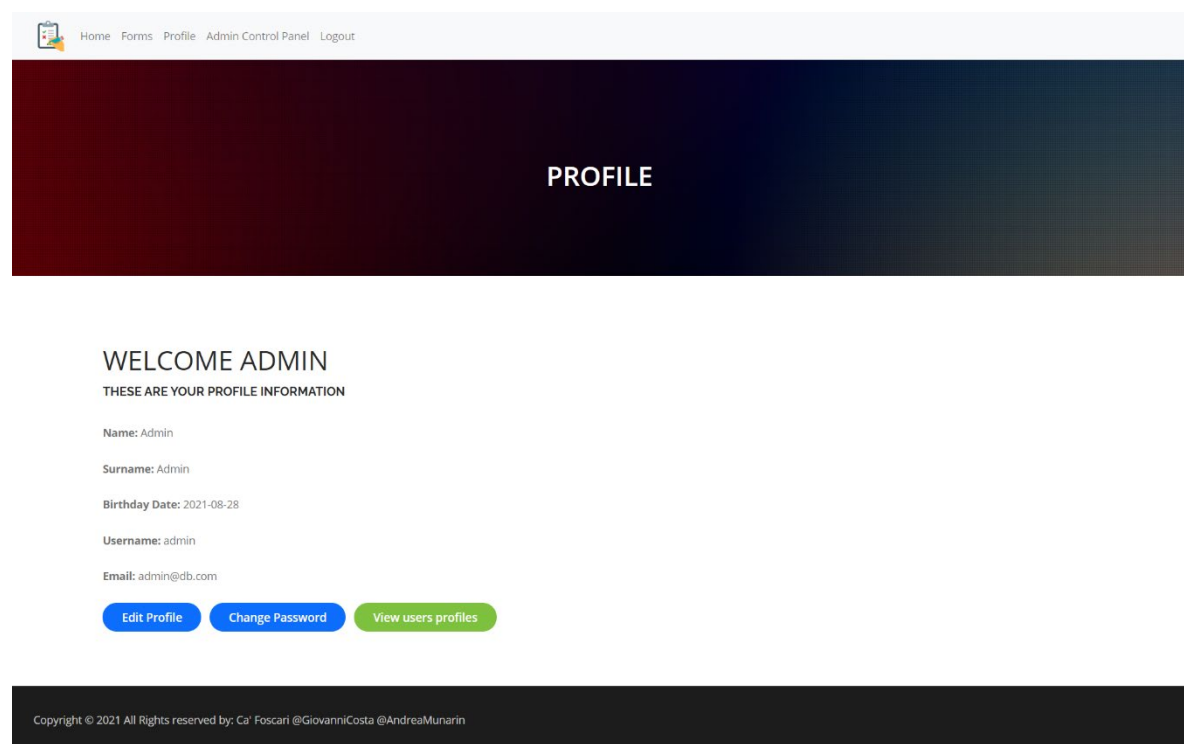
route: /<form_id>/add_question

2.5 Users

La gestione degli utenti è delegata quasi interamente a al framework Flask-security (ad esempio route per login, logout, registrazione, conferma del profilo tramite mail e cambio password).

Sono stati comunque forniti altri endpoint utili:

- `/profile` e `/profile/edit`
“`/profile`”, per permette di visualizzare tutte le informazioni riguardanti l’utente (ad esempio i dati personali) e per poter svolgere classiche azioni tipo la cancellazione del proprio account o il cambio password
“`/profile/edit`”, per modificare i relativi dati personali



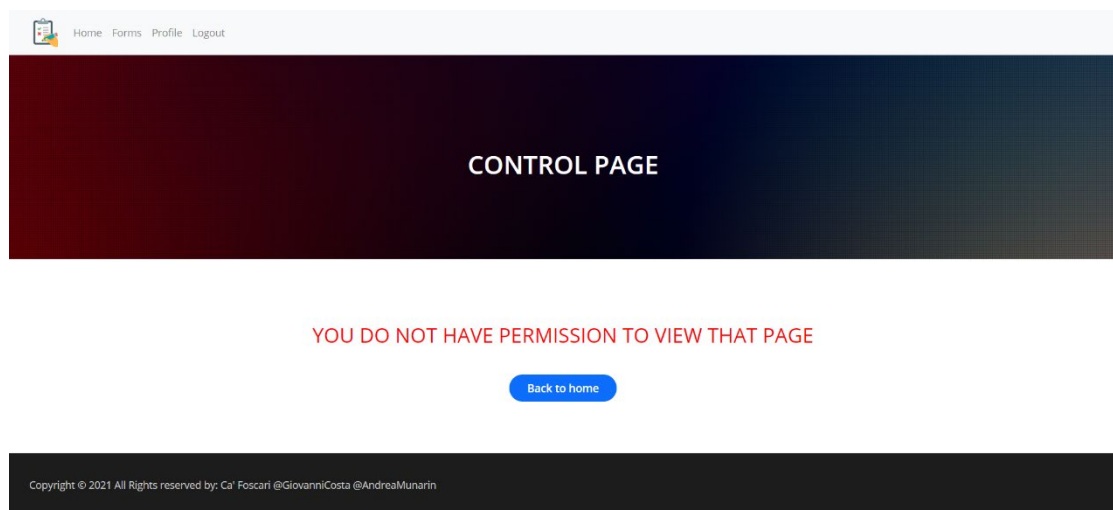
route: /profile

2.6 Error page

Error.html è una banale pagina html che sfrutta i template Jinja (un template engine) per visualizzare un messaggio variabile di errore a seconda dei casi

Questo messaggio, in particolare, è passato mediante la variabile Jinja “message”

Questa pagina è largamente da tutti gli endpoint: ad esempio, nel caso in cui venisse fatto un accesso errato alla route (come può essere se venisse specificato un form id che non esiste all'interno dell'indirizzo url) oppure nei casi in cui un inserimento/modifica non fosse andato a buon fine.



pagina di errore

3. Progettazione base di dati

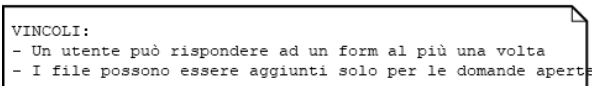
Si passa ora alla descrizione di come è stato modellato il problema della memorizzazione in un database dei vari dati della web application

3.1 Modello Concettuale

Database per la realizzazione di una web app per questionari online: Vari utenti registrati possono rispondere ai questionari e degli utenti si vuole conoscere l'e-mail, il nome, il cognome, l'età. Ad ogni utente sono associati dei ruoli che verranno utilizzati per le politiche di autorizzazione. Un questionario è identificato univocamente da un codice numerico e possiede un nome, una descrizione, la data di creazione, un utente creatore, delle domande e delle risposte. Ogni domanda è caratterizzata da una stringa che rappresenta la domanda e una serie di tag che ne descrive gli argomenti di appartenenza. Inoltre, ogni domanda può essere a risposta aperta, singola o multipla, dove queste ultime due avranno la possibilità di avere legate delle possibili risposte.

Le domande inserite in un determinato form possono essere obbligatorie, oppure possono permettere l'inserimento di un file come risposta, se queste domande sono a risposta aperta.

Le varie risposte invece sono caratterizzate da un utente che ha risposto, dal contenuto della risposta stessa (eventualmente appunto un file), dalla domanda alla quale fa riferimento, ed infine da che questionario proviene.



- Una domanda può avere più tag di appartenenza
- Le risposte a domande con risposta singola e multipla devono essere sequenze di stringhe
- Anche le possibili risposte associate alle domande con risposta singola e multipla devono essere sequenze di stringhe
- Le tre sottoclassi di domande sono utili a classificare il tipo di domanda creata
- Le varie risposte sono legate ad uno user: è necessario un trigger per evitare che un utente risponda due volte allo stesso form
- Sono necessari degli attributi sulla relazione tra forms e domande per riconoscere se questa è obbligatoria o se può contenere un file come risposta

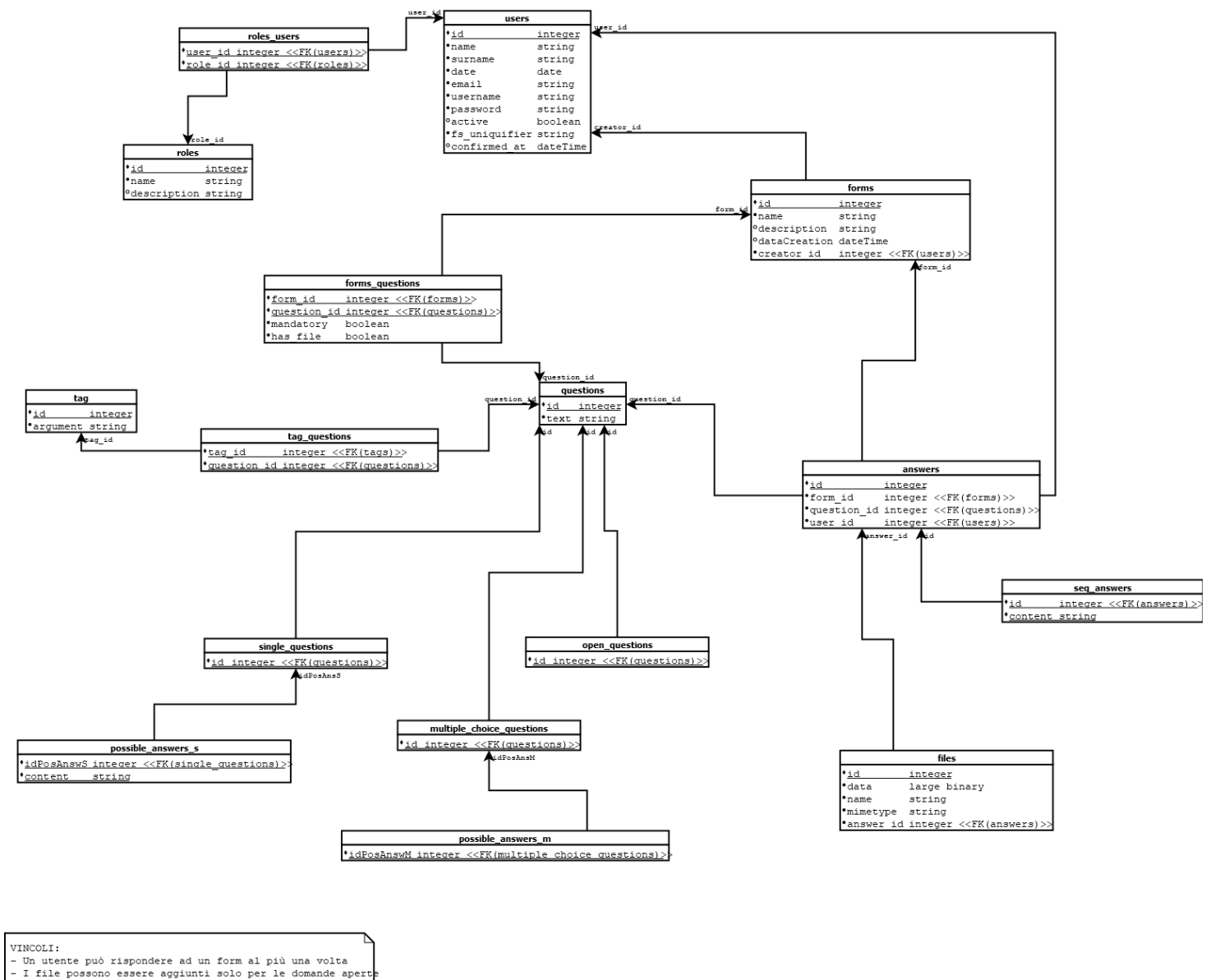
Il modello è stato progettato in modo che le domande vengano memorizzate nella base di dati solamente una volta, in modo da semplificare notevolmente la loro importazione in diversi form e ottimizzare il consumo di spazio nel database, visto che la maggior parte di questionari contiene domande simili tra tutti, tipo per quanto riguarda l'inserimento di dati personali come nome, cognome, e-mail.

Così facendo è necessario però svolgere particolari operazioni da effettuare quando viene deciso di modificare una domanda o di cancellarla

Per quanto riguarda la modifica è possibile usare una tecnica simile al Copy-on-write di alcuni sistemi operativi: la domanda viene “copiata” (nel database ne viene inserita una nuova, trascrivendo gli stessi attributi della precedente al meno di quelli “modificati”, che saranno sostituiti/aggiunti) ed infine il riferimento del form viene aggiornato alla nuova domanda

L'operazione di cancellazione vera e propria di una domanda non è invece possibile, perché essendo domande importabili e riutilizzate da molti, questa può portare a side effect eliminando la particolare domanda a tutti gli utenti che la riferiscono nel proprio form. Viene quindi rimosso solamente il riferimento “logico” tra domanda e form.

3.2 Modello Logico



I pallini vuoti di fianco agli attributi indicano che questi sono nullable, altrimenti sono not null

Traduzione da modello concettuale a logico, punti principali:

- Nelle chiavi esterne di: roles (solo su user_id), forms (solo su creator_id), forms_questions (solo su form_id), answers, file e seq_answers si è deciso di introdurre la politica di ON DELETE CASCADE per semplificare notevolmente le cancellazioni dei vari “gruppi logici” di elementi

- Si è scelto il partizionamento orizzontale per comodità e possibile utilità di impiego nelle relationship di SQLAlchemy ORM (Object Relational Mapping)
- Gli attributi in file come: name, data e mimetype rappresentano i dati e i metadati necessari per la memorizzazione dei documenti e per una loro successiva visualizzazione corretta
- Gli attributi in users come: fs_uniquifier e confirmed_at sono necessari per il corretto utilizzo di alcune funzionalità di Flask-Security

4. Query principali

Viene presentato ora un elenco delle query più utilizzate e/o le più complesse da comprendere

4.1 Possibili domande importabili

Per permettere all'utente di importare delle domande mediante dropdown list è necessario fornire al template Jinja la lista delle domande selezionabili (le domande dei template di base, le domande create dagli admin, le domande precedente inserite da quell'utente)

Questo viene fatto tramite 3 query "distinte", legate da due UNION

In particolare, le query sono:

1. Le domande nei form creati dall'utente corrente
2. Le domande nei form creati dagli Admin
3. Le domande di base caricate nel database

(Tutte le condizioni di giunzione sono specificate all'interno della clausola WHERE)

```
SELECT q.*
FROM questions q, formsQuestions fq, forms f
WHERE q.id = fq.question_id AND fq.form_id = f.id AND
f.creator_id = current_user.id
UNION
```

```

SELECT q.*
FROM questions q, forms_questions fq, forms f, users u, roles
r, rolesUsers ru
WHERE q.id = fq.question_id AND f.creator_id = u.id AND
fq.form_id = f.id AND r.id = ru.role_id AND u.id = ru.user_id
AND r.name = "Admin"
UNION
SELECT *
FROM questions
WHERE id > 0 AND id < 28

```

N.B.: `current_user` è una variabile di flask-security che riferenzia un'istanza dell'oggetto Users definita nel file "models.py"

4.2 Ricerca del form e delle domande di un form: le query più usate

Ricerca del form

```

SELECT *
FROM forms f WHERE f.id=form_id

```

Ricerca delle domande di un determinato form:

```

SELECT *
FROM forms_questions fq JOIN questions q ON
fq.question_id=q.id WHERE fq.form_id=form_id

```

4.3 Aggiunta e modifica delle domande di un form

Aggiunta di un nuovo Tag (se selezionato dalla pagina html)

```

INSERT INTO tag (argument) VALUES (new_tag_arg)

```

N.B.: `new_tag` è il valore che viene letto nel form html dalla dropdown list con multipla selezione

Aggiunta di una domanda

```

INSERT INTO questions (text) VALUES (text_questions)
INSERT INTO single_questions (id) VALUES (q.id)

```

```
for t in tag_list:
```

```
    INSERT INTO tags_questions (tag_id) VALUES (int(t))
```

- text_questions contiene il valore del testo della domanda ricavato dal form html
- q.id indica l'id, dove q fa riferimento alla domanda appena aggiunta
- Infine, vi è il "collegamento" mediante la relazione n a n convertita in tabella tra i tag relativi alla domanda (i cui id si trovano in una lista di stringhe ottenuta dalla dropdown list html) appena aggiunti e la domanda stessa

Aggiunta delle possibili risposte

```
for i in number:
```

```
    cont = Get value of form "i" (contenuto dell'i-esima  
    possible risposta)
```

```
    INSERT INTO possible_answersS (idPosAnsS, content) VALUES  
    (q.id, cont)
```

- number è il numero di possibili risposte scelte dall'utente in un text input html
- q anche qui fa riferimento alla domanda appena inserita

Aggiunta/modifica domanda nel form

I 3 precedenti step vengono effettuati sempre, quest'ultimo invece viene effettuato in base a se l'operazione scelta è una aggiunta, oppure se è una modifica

Aggiunta:

```
INSERT INTO forms_questions (form_id, question_id, mandatory,  
has_file) VALUES (form_id, q.id, mand, has_file)
```

- form_id è un valore che viene passato attraverso la route di flask
- mand (mandatory) e has_file (valido solo per le domande aperte) sono i valori di checkbox ricavati dal form html

Modifica:

Come descritto nel dettaglio prima, il database è stato progettato in modo che le domande vengano memorizzate solamente una volta, ma esse vengono riferite dai vari form se queste sono presenti al loro interno (tramite la tabella forms_questions), ad esempio nel caso una domanda venga importata

Nel caso si voglia quindi modificare una domanda è necessario creare ed inserirne una nuova utilizzando sia i parametri modificati, sia quelli rimasti invariati. Infine, occorre aggiornare il valore "question_id" nella tabella forms_questions al valore della nuova domanda

```
UPDATE forms_questions fq
WHERE fq.question_id = question_id
SET fq.question_id = q.id, fq.mandatory = mand
```

4.4 Visualizzazione delle risposte ad un form

Mediante questa query vengono mostrate le risposte relative ad un determinato form (tramite form_id dato dalla route di flask) nella pagina:

```
SELECT *
FROM answers a LEFT OUTER JOIN files f ON a.id = f.answer_id
WHERE a.form_id = form_id
```

N.B.: viene utilizzata una LEFT OUTER JOIN perché non tutte le domande hanno un file presente

Con questa invece vengono ricavati i dati sulle risposte relative ad un determinato form (tramite form_id dato dalla route di flask) per poi poterle passare come contenuto del file CSV scaricabile:

```
SELECT u.username, q.text, sq.content
FROM users u, questions q, answers a, seq_answers sq
WHERE a.form_id = form_id AND u.id = a.user_id AND a.id = sq.id AND q.id = a.question_id
```

4.5 Query di cancellazione dei form

Questa operazione è resa più agevole dal vincolo di ON DELETE CASCADE sulle chiavi esterne delle tabelle interessate, dove `f_id` è il parametro della route di flask

```
DELETE FROM forms f
WHERE f.id = f_id
```

N.B.: con questa banale istruzione vengono eliminate le righe corrette delle tabelle `forms`, `forms_questions`, `answers`, `seq_answers` e `files`

5. Principali scelte progettuali

Vediamo infine le principali scelte progettuali

5.1 Trigger definiti

Abbiamo implementato i seguenti due trigger:

- Il trigger “`only_one_answer`” controlla che non vengano inserite risposte a domande di un form già compilata, in caso contrario l'operazione viene annullata

```
CREATE TRIGGER only_one_answer BEFORE INSERT ON answers
FOR EACH ROW EXECUTE PROCEDURE only_one_answer_func();

CREATE FUNCTION only_one_answer_func()
RETURNS TRIGGER AS $$
BEGIN
IF (NEW.user_id, NEW.question_id) IN (SELECT user_id,
question_id FROM answers WHERE form_id=NEW.form_id) THEN
RETURN NULL;
END IF;
RETURN NEW;
END; $$ LANGUAGE PLPGSQL;
```

- Il trigger “only_onOpenQuestion” si accerta invece che se in una domanda è possibile aggiungere un file, essa deve essere una domanda a risposta aperta, in caso contrario l’operazione viene annullata

```
CREATE TRIGGER only_onOpenQuestion BEFORE INSERT ON
forms_questions
FOR EACH ROW EXECUTE PROCEDURE
file_only_onOpenQuestion_func();

CREATE FUNCTION file_only_onOpenQuestion_func()
RETURNS TRIGGER AS $$
BEGIN
IF NEW.has_file AND ( (NEW.question_id IN (SELECT s.id FROM
single_questions s)) OR (NEW.question_id IN (SELECT m.id FROM
multiple_choice_questions m)) ) THEN
RETURN NULL;
END IF;
RETURN NEW;
END; $$ LANGUAGE PLPGSQL;
```

Nonostante la web app sia protetta con controlli appropriati nelle route, che quindi non permettano di effettuare questi inserimenti nel database, per questioni di sicurezza abbiamo deciso di implementare comunque questi trigger in modo da salvaguardare ulteriormente la base di dati da anomalie

5.2 Sicurezza: Ruoli e prevenzione da SQL injection

Ecco i ruoli creati con le relative funzionalità:

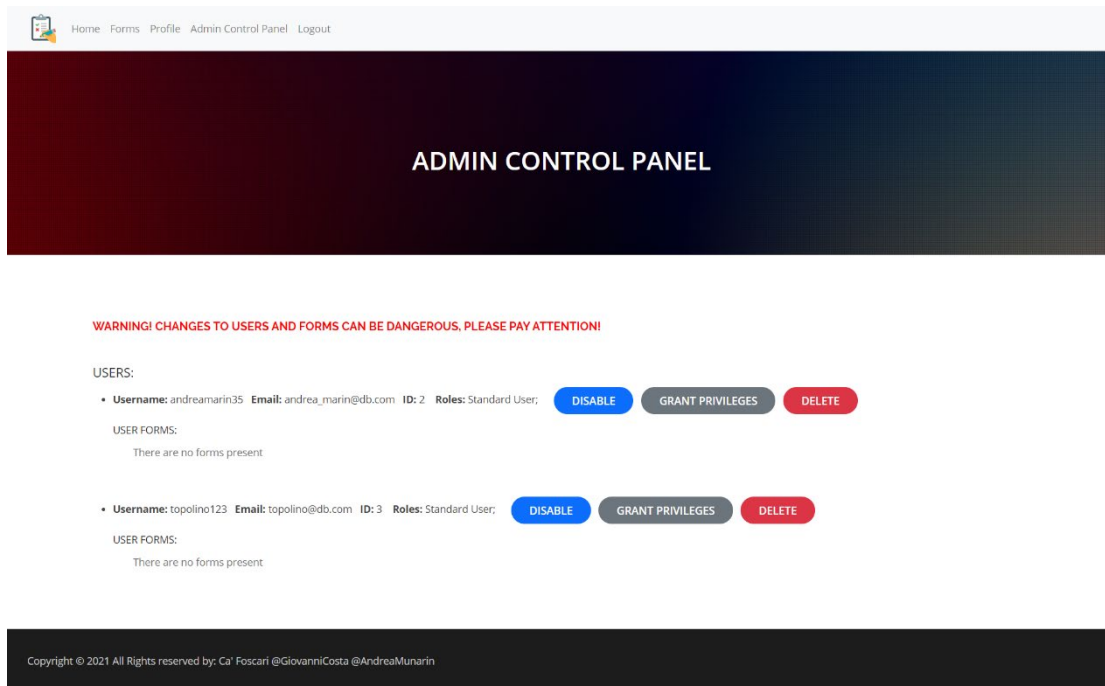
- SuperUser
 - Garantire e revocare il ruolo di admin
 - Modificare/eliminare form di utenti admin
 - Cancellare account admin
 - Disabilitare account admin
 - Tutti i permessi del ruolo Admin
- Admin:
 - Modificare/eliminare form di utenti Standard User
 - Cancellare account Standard User
 - Disabilitare account Standard User
- Standard User
 - Nessun permesso particolare
 - Può interfacciarsi con le funzionalità principali della web app

La maggior parte degli endpoint sono marcati col decoratore di Flask-Security “@auth_required()” che rende necessario effettuare il login o la registrazione all’app prima di raggiungere l’indirizzo URL desiderato

Inoltre, per garantire il controllo degli accessi, sono stati creati dei decoratori ad-hoc con cui sono stati marcati diverse route, quali:

- @creator_or_admin_role_required
- @admin_role_required
- @superuser_role_required

Gli utenti Admin e SuperUser dispongono inoltre di un’interfaccia specifica per svolgere operazioni di amministrazione sugli utenti (congrua con i privilegi definiti prima) accessibile dal pulsante “View users info” all’endpoint “/profile”, oppure tramite la navbar usando il link “Admin Control Panel”



Interfaccia di amministrazione per un SuperUser (simile a quella per un Admin)

Il problema dell'SQL Injection viene risolto in automatico usando SQLAlchemy ORM che svolge nativamente una sanitizzazione degli input dati in ingresso alle query e da Flask tramite l'utilizzo dei template Jinja che svolgono l'escaping delle variabili passate

5.3 Performance

La ricerca delle domande in base al form corrente e la visualizzazione delle risposte legate ad uno specifico form sono due delle operazioni più comuni all'interno del database, per questo motivo si è pensato di creare due materialized view che fossero la memorizzazione del risultato di una giunzione tra “forms” e “questions”, e “forms” e “answers”, rispettivamente

```
CREATE MATERIALIZED VIEW f_questions AS SELECT questions.*
FROM questions, forms_questions, forms
WHERE forms_questions.question_id = questions.id AND
forms_questions.form_id = forms.id
```

```
CREATE MATERIALIZED VIEW f_answers AS SELECT answers.* FROM  
answers, forms WHERE answers.form_id = forms.id
```

Come visto a lezione quindi, il query planner sfrutterà in automatico queste viste materializzate contenenti dati già calcolati per ottimizzare le varie query e migliorarne le performance generali, a discapito di una periodica manutenzione delle due view per mantenerle aggiornate

Non sono stati dichiarati indici nelle tabelle perché PostgreSQL, il DBMS utilizzato, esegue in automatico la creazione di indici su chiavi primarie e chiavi esterne e questi sono stati considerati sufficienti

5.4 Utilizzo delle transazioni e ORM

Viene scelta come tecnica di programmazione l'Object-Relational Mapping perché offre consistenti vantaggi rispetto all'utilizzo di API per l'interfacciamento al SQL:

- Indipendenza dallo specifico DBMS sottostante
- Non richiede conoscenza approfondita di SQL
- Migliore supporto da parte del compilatore
- Astrazione da dettagli di basso livello (es. sanitizzazione)
- Mitigazione dei problemi legati all'impedance mismatch

Le transazioni sono state settate globalmente all'instaurazione della connessione col database su livello di isolamento "REPEATABLE READ", che impedisce il fenomeno delle dirty read, garantendo ad una transazione che vuole effettuare una scrittura un lock, il quale viene rilasciato solo dopo la sua terminazione

Questo livello di isolamento è considerato sufficiente per la web application in questione perché fenomeni come unrepeatable read e lost update non creano importanti problemi e non generano inconsistenze di dati

6. Librerie necessarie per il funzionamento dell'app

```
pip install flask-security-too sqlalchemy bcrypt
pip install flask_babelex
pip install Flask-Mail
pip install python-dotenv
pip install psycopg2      //se si usa come DBMS postgresQL
```