Documentazione progetto LoHacker

Gruppo: LoHacker

Membri: Castellani Mattia (881427), Pastrello Marco (881679), Pellizzon Tommaso (880772)

Progetto: Creazione di questionari Online

Introduzione

Il progetto consisteva nel creare una web application che si interfacciava con una base di dati utilizzando il linguaggio Python e due librerie: Flash e SQLAlchemy. L'applicazione da noi creata permette la creazione e la gestione di questionari online. Per lo sviluppo abbiamo cercato di immaginarci quali potrebbero essere le funzionalità più utili per un utente mantenendo un design facile da capire, inoltre ci siamo basati molto anche su Google Form cercando di capire quali potrebbero essere le funzionalità più interessanti da inserire nel nostro progetto.

Funzionalità principali

L'applicazione permette la registrazione o il log in di un utente, dopo di che quest'ultimo può o creare un form o rispondere ad uno di essi.

Quando un utente decide di creare un form ha inizialmente due opzioni principali: creare un form direttamente partendo con una base vuota o crearlo partendo da un template creato in precedenza (la creazione del template è come quella di un normale form tuttavia non viene reso accessibile la possibilità di risponderci), qui l'utente dovrà inserire il titolo del form e cominciare ad inserire le domande di quattro possibili: domande aperte, domande checkbox, radio button (con la possibilità di incrementare il numero di opzioni e di modificarne il testo) e domande file (dove è necessario l'inserimento di un file).

Tutte le domande hanno i campi dove è possibile inserire il testo di ogni domanda o opzione, in più è possibile decidere se il form debba essere anonimo o meno e quali (se ce ne sono) debbano essere le domande obbligatorie che richiedono una risposta da parte dell'utente.

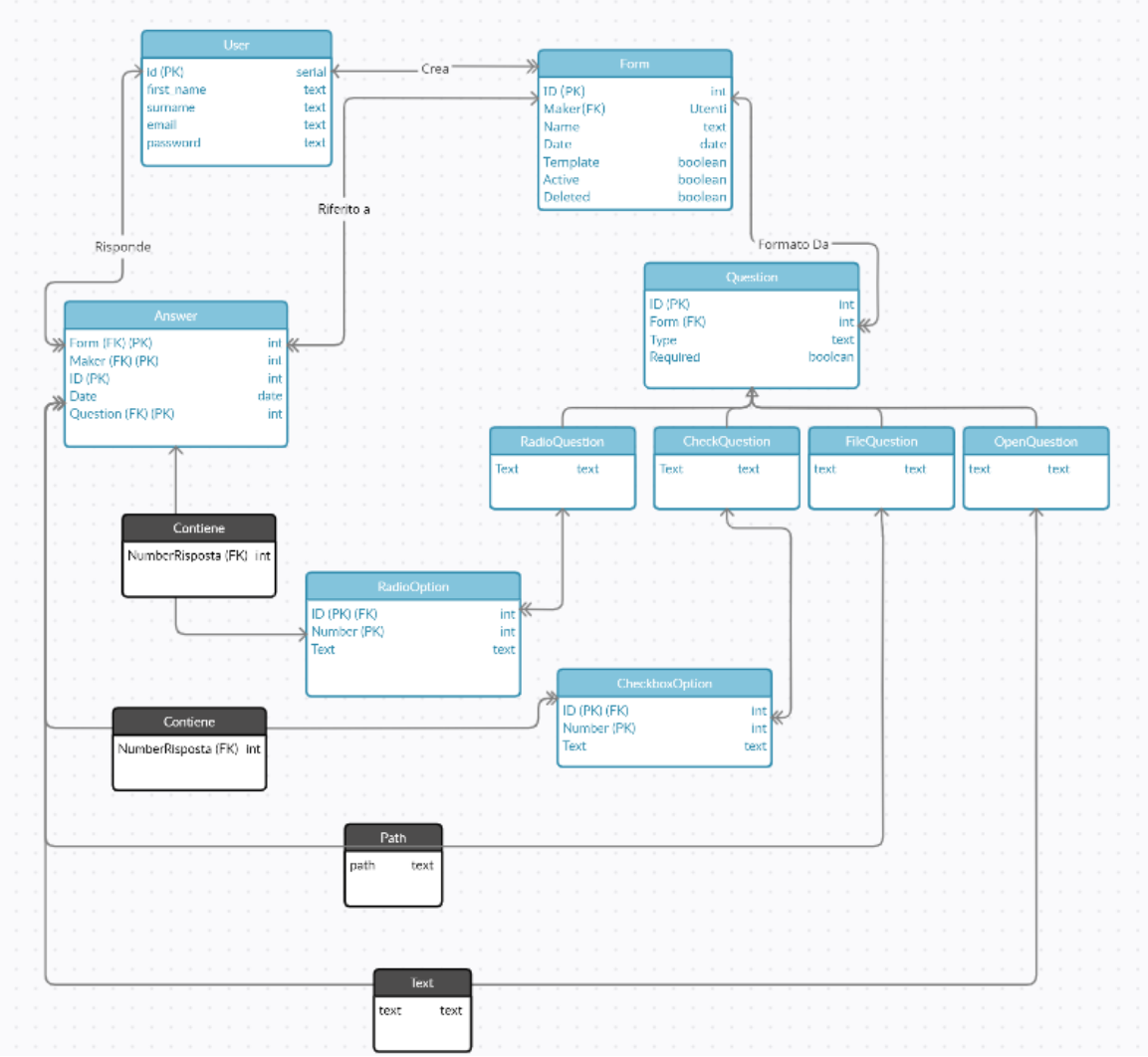
Terminata la creazione viene condiviso un link o l'id del questionario da condividere con l'utente che dovrà dare una risposta, successivamente saranno visualizzabili le risposte per ogni questionario per singolo utente o in generale per ogni domanda e sarà possibile l'eliminazione del form o la disattivazione di quest'ultimo. Nel caso in cui il form venga disattivato dal proprietario, allora se un utente proverà a rispondere al form verrà visualizzato un messaggio d'errore. Inoltre è stata realizzata la funzionalità che permette di scaricare il file csv relativo al form che contiene le domande e per ognuna di queste le risposte che sono state inviate.

Per rispondere ad un questionario sarà sufficiente inserire l'id o il link, ottenuto dal proprietario, per poter rispondere, sarà anche possibile visualizzare se il form a cui si risponde è anonima o meno. In particolare il gruppo ha deciso che un utente che ha creato un form può anch'esso rispondere al form. Inoltre è stato deciso che un utente può rispondere ad un form una sola volta, nel caso siano fatti più tentativi sarà visualizzato un messaggio di errore.

Progettazione concettuale e logica

Abbiamo cercato fin da subito di pensare ad una base di dati il quanto più possibile vicina a quella che potrebbe essere la versione finale ed è qui che abbiamo riscontrato alcuni problemi nel creare un database SQL per questa applicazione, ad esempio: il salvataggio delle risposte, la creazione dei template e la struttura logica delle domande. Quindi dopo la prima parziale risoluzione dei problemi abbiamo provato a creare una struttura logica di partenza per poter almeno iniziare a capire bene quali potrebbero essere altri problemi o altre soluzioni più corrette.

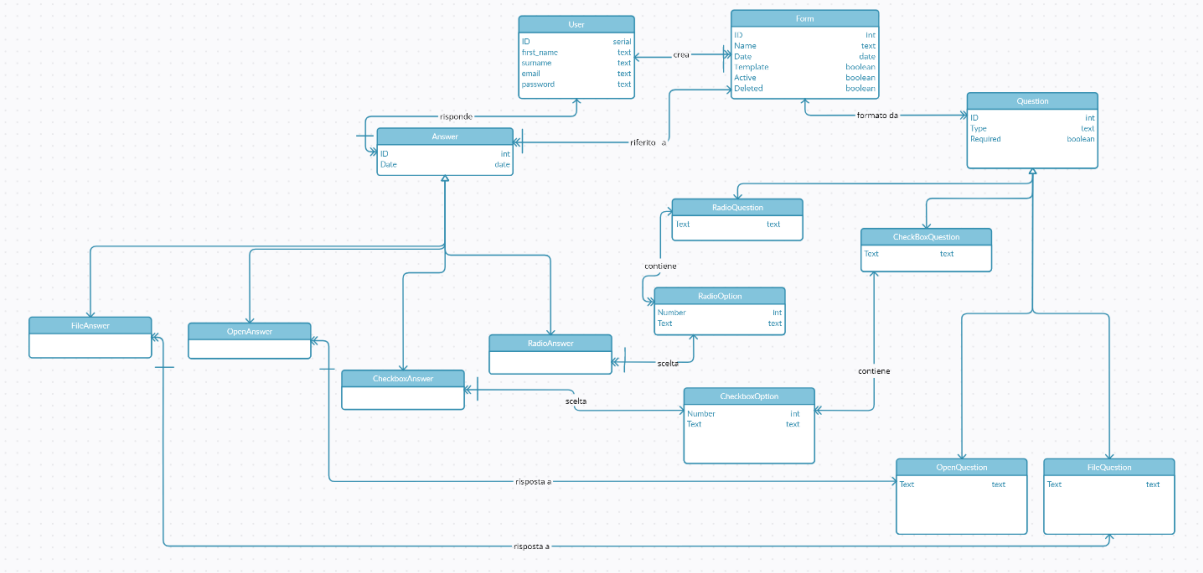
Come primo approccio abbiamo pensato alle entità principale che potrebbero essere presenti come: utente, form, domande e risposte. Dopo di che abbiamo cercato di differenziare le domande in più tipi (come checkbox e open answer) ricordando che i tipi checkbox e radio al loro interno contengono le varie opzioni di risposta alla domanda. Qui sotto è allegato l'immagine che rappresenta uno schema del nostro primo pensiero dopo i primi giorni dove abbiamo cercato di mantenerci il più corretti possibile rispetto ad anomalie e normalizzazione preferendo però alcune volte la facilità di implementazione.



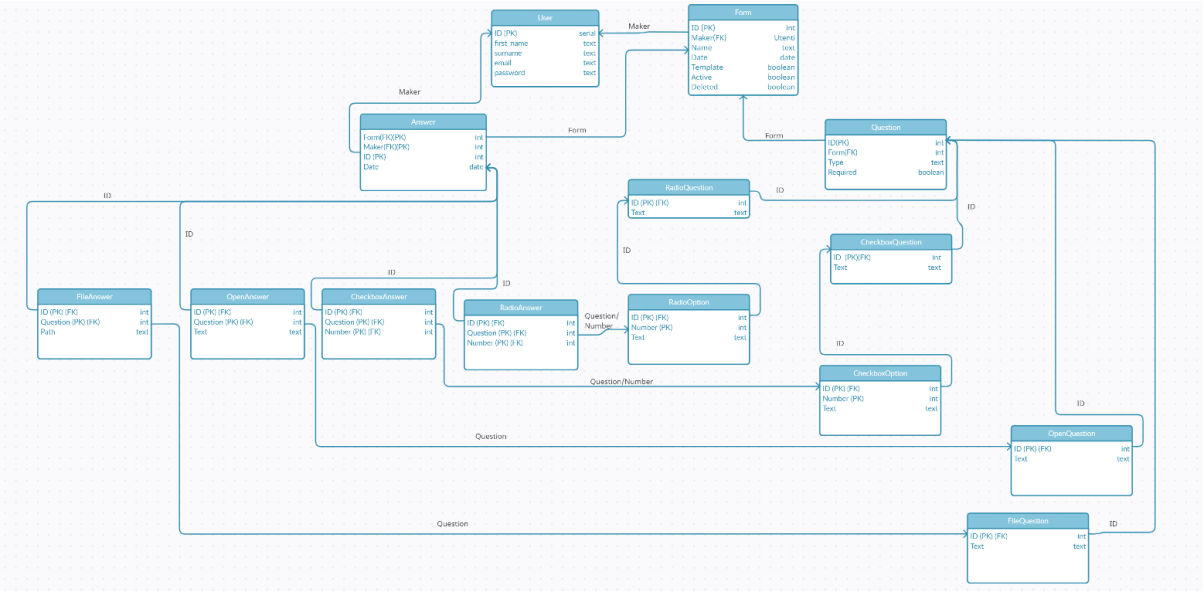
Si può notare come sia presente una ereditarietà dell'entità Question per poter differenziare le varie tipologie, mentre per Answer è presente un'associazione con il form per poter facilitare e velocizzare le query e l'implementazione. Abbiamo deciso di inserire un id in user come chiave primaria e (anche se non è segnato nello schema) email come unique, successivamente abbiamo dovuto prendere decisioni mirate a trasformare il prototipo iniziale dello schema in una versione sempre più finale, completa e corretta, il risultato ottenuto è allegato qui sotto.

Di seguito alleghiamo il modello concettuale e successivamente il modello logico.

Schema concettuale



Schema logico



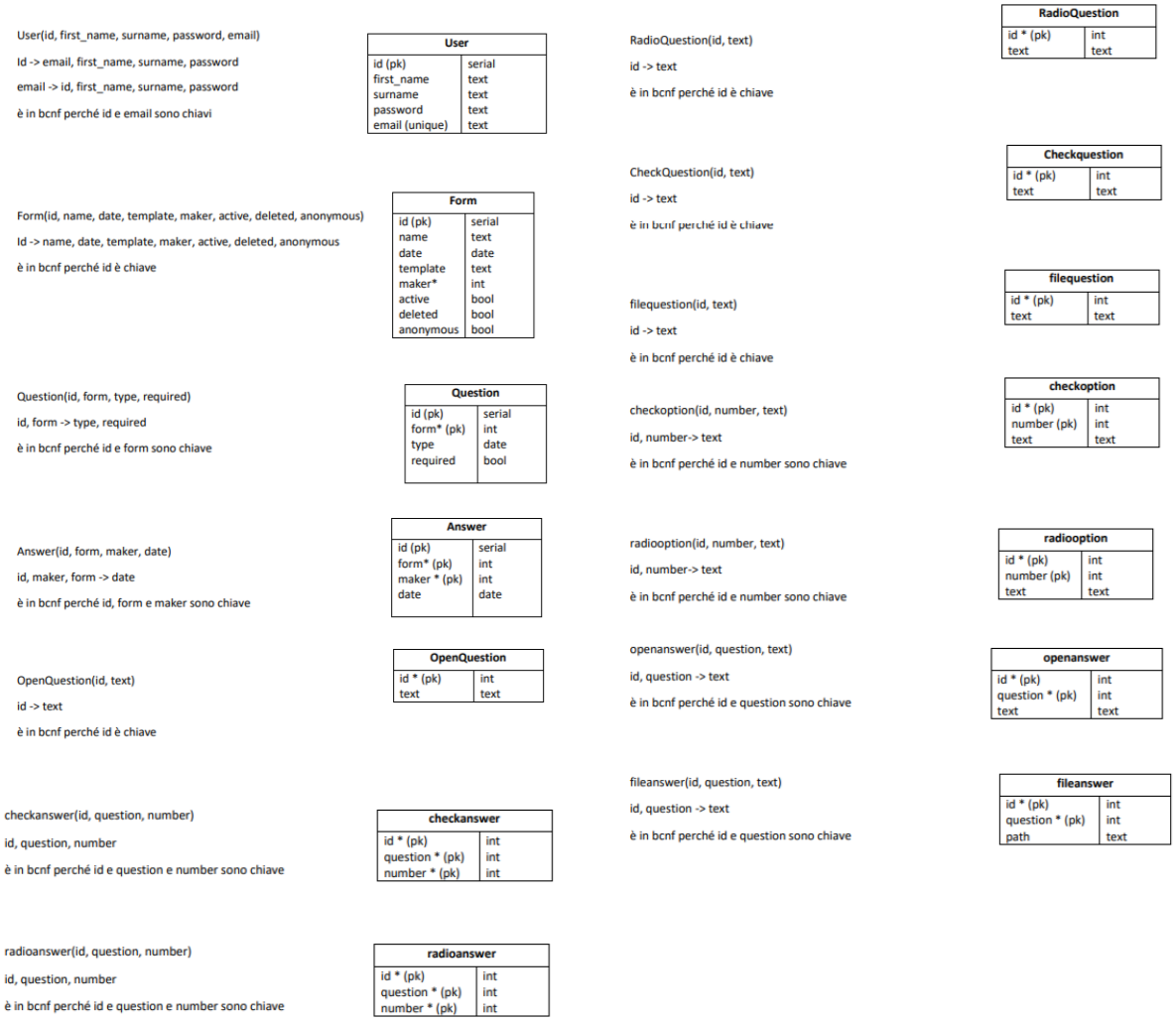
Come è visibile abbiamo prima di tutto deciso di implementare l'ereditarietà in maniera verticale inserendo RadioQuestion, CheckboxQuestion, FileQuestion e OpenQuestion associate all'entità Question dove sono presenti tutti gli attributi comuni, mentre nelle singole entità è presente il testo della domanda, la motivazione di questa scelta, invece di inserire il testo come attributo dell'entità Question, è principalmente per fare una distinzione chiara delle varie tipologie di domande, in modo tale che se in versioni successive una precisa tipologia di domanda necessita di modifiche specifiche, è possibile realizzarle senza andare ad intaccare la struttura delle altre domande, garantendo la retro compatibilità. Inoltre su Question è presente l'attributo type che per una scelta puramente implementativa abbiamo deciso di inserirlo facilitando il processo di ricerca e inserimento delle domande. Abbiamo deciso di fare lo stesso processo dell'ereditarietà verticale per Answer in moda da poter mantenere chiarezza e simmetria rispetto Question e in più qui si avranno diversi attributi tra entità fratelle, essendoci l'attributo "number" per le CheckboxAnswer e le RadioAnswer, un path (che permette di memorizzare il percorso del file di riferimento all'interno del server) per le FileAnswer e il testo per le OpenAnswer.

Per quanto riguarda gli identificatori delle varie entità (attributo id) è stato scelto di affidare l'assegnamento del valore al database, in modo da evitare qualsiasi equivoco, quindi questi attributi sono di tipo SERIAL. Questo è realizzato grazie alle sequenze, quando si assegna il valore della sequenza ad un id, il valore della sequenza viene incrementato di una unità.

Risulta necessaria una considerazione riguardo all'associazione tra Answer e Form: dal momento che il Form di riferimento di una Answer può essere recuperato utilizzando una delle specifiche tipologie di Answer, come OpenAnswer, FileAnswer ecc. le quali sono collegate ad una specifica domanda che è contenuta in un Form, allora si potrebbe concludere che tale

collegamento è inutile. Tuttavia l'implementazione di questo collegamento tra le due entità risulta necessario per evitare lo scenario successivamente proposto. Lo scenario che si vuole evitare è quello in cui un utente risponde ad un form composto da domande che non sono obbligatorie. Allora la risposta dell'utente potrebbe essere vuota, o più precisamente, una risposta che però non va ad inserire alcuna row all'interno delle varie TypeAnswer. In tal caso non sarebbe più verificabile se un utente ha già risposto, anche se con una risposta vuota, ad un specifico form.

Questa è la nostra verifica del mantenimento della Boyce Codd Normal Form.



Query principali

Selezioniamo alcune delle query più importanti all'interno del progetto per una analisi più approfondita:

Recupero opzioni selezionate nelle risposte delle domande di tipo checkbox e radio

Recupero delle opzioni delle checkbox selezionate nelle risposte

In Python usando l'ORM di SQLAlchemy

```
session.query(CheckboxOption.id, CheckboxOption.number, func.count(CheckboxAnswer.number).label('counter'))
            .outerjoin(CheckboxAnswer,(CheckboxAnswer.question == CheckboxOption.id)&(CheckboxAnswer.number==CheckboxOption.number))
            .filter(CheckboxOption.id == questionID).group_by(CheckboxOption.id, CheckboxOption.number).all()
```

Traduzione in SQL

```
SELECT CheckboxOption.id, CheckboxOption.number, COUNT(CheckboxAnswer.number) AS 'counter'
FROM CheckboxAnswer LEFT JOIN CheckboxOption ON CheckboxAnswer.question = CheckboxOption.id AND CheckboxAnswer.number =CheckboxOption.
WHERE CheckboxOption.id = questionID
GROUP BY CheckboxOption.id, CheckboxOption.number
```

Recupero delle opzioni delle checkbox selezionate nelle risposte

In Python usando l'ORM di SQLAlchemy

```
session.query(RadioOption.id, RadioOption.number, func.count(RadioAnswer.number).label('counter'))
            .outerjoin(RadioAnswer, (RadioAnswer.question == RadioOption.id)&(RadioAnswer.number == RadioOption.number))
            .filter(RadioOption.id == questionID).group_by(RadioOption.id, RadioOption.number).all()
```

Traduzione in SQL

```
SELECT RadioOption.id, RadioOption.number, COUNT(RadioAnswer.number) AS 'counter'
FROM RadioAnswer LEFT JOIN RadioOption ON RadioAnswer.question = RadioOption.id AND RadioAnswer.number = RadioOption.number
WHERE RadioOption.id = questionID
GROUP BY RadioOption.id, RadioOption.number
```

Queste query molto simili, dato l'ID del form, permette di recuperare per ogni domanda di tipo checkbox o radio il numero di opzioni che sono state selezionate in tutte le risposte al relativo form. Il counter permette appunto di contare le occorrenze delle opzioni selezionate, e necessita di una giunzione di tipo `LEFT OUTER JOIN` perché così facendo anche le opzioni che non sono state selezionate in alcuna risposta vengono ritornate con un occorrenza pari a zero.

Recupero di tutte le domande di un form

```
def getAllFormQuestion(formID):
  questionsSet = []
  questionForm = session.query(Question).filter(Question.form == formID).order_by(Question.id).all()

  for question in questionForm:
       if question.type == "open":
          questionSet += getOpenQuestion(question.id)
       elif question.type == "checkbox":
          questionSet += getCheckboxQuestion(question.id)
       elif question.type == "radio":
          questionSet += getRadioQuestion(question.id)
       elif question.type == "file":
          questionSet += getFileQuestion(question.id)
  return questionSet
```

Funzione `getOpenQuestion(questionID)`

```
session.query(Question.type, OpenQuestion.id, OpenQuestion.text, Question.required).join(OpenQuestion).filter(Question.id == questionI
```

Traduzione della query in SQL

```
SELECT Question.type, OpenQuestion.id, OpenQuestion.text, Question.required
FROM Question NATURAL JOIN OpenQuestion
WHERE Question.id = questionID
```

Funzione `getCheckboxQuestion(questionID)`

```
session.query(Question.type, CheckboxQuestion.id, CheckboxQuestion.text, Question.required).join(CheckboxQuestion).filter(Question.id
```

Traduzione della query in SQL

```
SELECT Question.type, CheckboxQuestion.id, CheckboxQuestion.text, Question.required
FROM Question NATURAL JOIN CheckboxQuestion
WHERE Question.id = questionID
```

Funzione `getRadioQuestion(questionID)`

```
session.query(Question.type, RadioQuestion.id, RadioQuestion.text, Question.required).join(RadioQuestion).filter(Question.id == questi
```

Traduzione della query in SQL

```
SELECT Question.type, RadioQuestion.id, RadioQuestion.text, Question.required
FROM Question NATURAL JOIN RadioQuestion
WHERE Question.id = questionID
```

Funzione `getFileQuestion(questionID)`

```
session.query(Question.type, FileQuestion.id, FileQuestion.text, Question.required).join(FileQuestion).filter(Question.id == questionI
```

Traduzione della query in SQL

```
SELECT Question.type, FileQuestion.id, FileQuestion.text, Question.required
FROM Question NATURAL JOIN FileQuestion
WHERE Question.id = questionID
```

La funzione `getAllFormQuestion(formID)` è una delle funzione più importanti utilizzate all'interno dell'applicazione web. Questa permette di ottenere tutte le domande, di qualsiasi tipo, che appartengono ad un form. In particolar modo inizialmente vengono cercate tutte le domande, di tipo generico appartenenti a Question, ordinandole in modo crescente secondo Question.id. Questo passaggio permetterà di costruire il set delle domande nello stesso ordine con il quale il form e le sue domande sono stati creati. Successivamente andando a ispezionare il tipo di ogni Question si va a ricavare la precisa tipologia di domanda, inserendola poi in `questionSet`, una lista che terminate tutte le iterazioni conterrà tutte le domande, specifiche di ogni tipo, del questionario.

Questa funzione viene utilizzata ad esempio per recuperare le domande da stampare nella pagina per permettere all'utente di rispondere, oppure per recuperare le domande di un template per poi andarle a stampare nella pagina di creazione del form.

Creazione di un form

```
form = Form(maker = current_user.get_id(), name = formTitle, date = date.today(), template = False, active = True, deleted = False, an
session.add(form)
session.commit()
```

Traduzione in linguaggio SQL

```
INSERT INTO Form(maker, name, date, template, active, deleted, anonymous)
VALUES (current_user.get_id(), formTitle, date.today(), False, True, False, anonymousOption)
```

La query qui sopra riportata è una query di tipo INSERT che permette di creare un form.

L'id del form non è necessario che venga specificato dal momento che l'attributo è di tipo seriale e quindi l'assegnamento di default è effettuato dal database.

Inserimento delle domande

```
for k,v in formRequest:
    k = k.split()[1]
    if k == 'open':
       newQuestion = Question(form = str(form.id), type = "open", required = False)
       session.add(newQuestion)
       session.commit()
       newOpenQuestion = OpenQuestion(id = str(newQuestion.id), text = v)
       session.add(newOpenQuestion)
       idRequiredQuestion = newQuestion.id
```

```
    elif k == 'checkbox':
       newQuestion = Question(form = str(form.id), type = "checkbox", required = False)
       session.add(newQuestion)
       session.commit()
       newCheckboxQuestion = CheckboxQuestion(id = str(newQuestion.id), text= v)
       session.add(newCheckboxQuestion)
       id_check = newCheckboxQuestion.id
       idRequiredQuestion = newQuestion.id
       i = 1
    elif k == 'checkboxtext':
        newCheckboxOption = CheckboxOption(id = str(id_check), number = i, text = v)
        session.add(newCheckboxOption)
        i = i + 1
    elif k == 'radio' :
         newQuestion = Question(form = str(form.id, type = "radio", required = False)
         session.add(newQuestion)
         session.commit()
         newRadioQuestion = RadioQuestion(id = str(newQuestion.id), text= v)
         session.add(newRadioQuestion)
         id_radio = newRadioQuestion.id)
         idRequiredQuestion = newQuestion.id
         i = 1
     elif k == 'radiobtntext' :
        newRadioOption = RadioOption(id = str(id_radio), number = i, text = v)
        session.add(newRadioOption)
        i = i + 1
     elif k == 'fileText':
        newQuestion = Question(form = str(form.id), type = "file", required = False)
        session.add(newQuestion)
        session.commit()
        newFileQuestion = FileQuestion(id = str(newQuestion.id), text = v)
        session.add(newFileQuestion)
        idRequiredQuestion = newFileQuestion.id
     elif k == "required":
        requiredQuestion = session.query(Question).filter(Question.id) == idRequiredQuestion).first()
        requiredQuestion.required = True
        session.commit()
    session.commit()
```

Il segmento di codice qui sopra riportato permette di creare e inserire tutte le domande, generiche e di ogni tipo, all'interno del Database. I dati da inserire sono contenuti all'interno della lista `request.form[]` la quale viene iterata. I nomi dei campi di input delle domande sono strutturati in modo tale che nella prima parte sia contenuto il tipo della domanda e nella seconda un numero sequenziale che ne permette di distinguere l'ordine. In base al tipo viene quindi prima creata la domanda generica, ovvero Question e successivamente quella più specifica che varia in base al tipo specificato nella chiave ottenuta dall'iterazione di `request.form[]`. Nel caso di domande di tipo checkbox e di tipo radio è necessario anche aggiungere le varie opzioni. Anche in questo caso il nome del campo di input segue una convenzione che permette di identificare di che genere di opzione si tratta. Infine si può osservare che come ultimo ramo dell' `if-then-elif-else` si trova il "required" che permette di rendere una domanda obbligatoria. Ogni volta che una domanda viene creata ne viene salvato l'id in una variabile, se in un iterazione la chiave è uguale a "required" allora l'ultima domanda inserita con id memorizzato nella variabile `idRequiredQuestion` verrà modificata, eseguendo una query di update che va a modificare l'attributo required della domanda da `False`, valore di default, a `True`.

Eliminazione di un form

```
def deleteForm(form):
  form.deleted = True
  session.commit()
```

Traduzione in linguaggio SQL

```
UPDATE Form
SET deleted = True
WHERE id = formID
```

La query qui presentata permette di eliminare un form dalla base di dati. Il gruppo ha scelto di adottare un'eliminazione logica, evitando quindi di perdere dati che per un amministratore della base potrebbe comunque essere necessari. Quindi nella base di dati non avviene alcuna operazione di `DELETE` bensì qualsiasi genere di eliminazione adotta un implementazione di tipo logico, andando così a modificare semplicemente un attributo che indica se il Form è eliminato o meno. Quando una query dovrà andare a recuperare uno o più form, sarà suo compito verificare che questo/i non sia eliminato.

Creazione di una risposta con inserimento di un file

```
def createNewFileAnswer(value, questionID, answerID):
  name = value + ' file'
  file = request.files.get(name)
  if file.filename != "":
    extension = file.filename.split(".")[1]
    newName = "fileQ"+str(questionID)+"A"+str(answerID)+"."+extension
    newFileAnswer = FileAnswer(id = str(answerID), question = questionID, path = newName)
    session.add(newFileAnswer)
    session.commit()
    file.filename = newName
    file.save(os.path.join(app.config['UPLOAD_FOLDER'], secure_filename(file.filename)))
```

Questa funzione permette di andare a salvare un file nel server. La scelta del gruppo è stata quella di memorizzare i file nel blob e andare a salvare nella base di dati il nome del file che viene memorizzato invece di usare attributo di tipo blob, in grado di memorizzare un file, ma con performance per i tempi di modifica e lettura maggiori, da qui la scelta di utilizzare la tecnica di salvataggio nel server.

La funzione, dopo una prima fase di costruzione del nome del file in modo che questo sia sempre univoco, prima crea la entry da inserire nella tabella `FileAnswer`, successivamente tramite il metodo `file.save()` salva il file (che prima ne viene controllata la sicurezza del filename tramite la `funzione secure_filename()`) nel path specificato come primo parametro.

Principali scelte progettuali

Le prime scelte che sono state fatte dal gruppo riguardavano inizialmente in che modo e con quale linguaggio l'applicazione dovesse essere implementata. A tal proposito abbiamo scelto di utilizzare la libreria di Python SqlAlchemy ORM, molto più semplice e pratica rispetto a SqlAlchemy Core.

Successivamente, in fase di sviluppo, è stato notato e deciso che utilizza le Blueprint avrebbe strutturato meglio il lavoro, potendo così separare i vari endpoint secondo un senso logico, in base alla funzione e al contesto di appartenenza. Eccezion fatta per il file `connection.py`: file che non appartiene ad alcun contesto applicativo e che non necessita quindi di dover appartenere ad una precisa sezione della struttura. Talvolta è un file globale, che contiene la connessione al database, i vari data model, numerose funzioni che sono usate più volte e le funzioni che contengono le query.

Infine per quanto riguarda la parte view ci siamo affidati a jinja che grazie all'ereditarietà ci ha permessi di creare una pagina di base, la quale viene eredita da tutte le altre pagine che andranno poi a riempirne il corpo di questa pagina. Jinja in questo caso è risultato molto utile poiché siamo riusciti a dare uno stesso schema a tutte le pagine. Inoltre jinja è risultato molto utile dal momento che permette di fare iterazioni e controlli sui parametri passati dal codice python.

Le successive scelte che andremo a spiegare sono decisioni mirate a garantire l'integrità dei dati e dell'intera applicazione.

Check constraint

Controllo type Question

```
((type = 'open'::text) OR (type = 'checkbox'::text) OR (type = 'radio'::text) OR (type = 'file'::text))
```

Il check qui presente è stato necessario per controllare che l'attributo type potesse assumere solo determinati valori. Nel caso ciò non avvenisse l'applicazione non funzionerebbe poiché determinati servizi, come la stampa delle domande di un form, si basano sulle varie tipologie delle domande, infatti in base al tipo verranno eseguite azioni diverse, ad esempio se una domanda è di tipo checkbox allora sarà necessario andare a recuperare anche le relative opzioni, per una domanda aperta ciò non è necessario.

Controllo validità template

Questo controllo è mirato a verificare che, se un form che viene inserito è un template (attributo `template = true`), allora questo deve essere sempre attivo, attributo `active = true`, e che il template non sia anonimo, attributo `anonymous = false`. Per realizzare questi controlli, che si presentano sotto forma di implicazioni, abbiamo usato la trasformazione di implicazioni in operazioni booleane. In particolare:

```
A => B  :  !A OR B
```

Quindi il controllo che se abbiamo un template allora `active` deve essere true si traduce con

```
template = true => active = true  :  template = false OR active = true
```

Per quanto riguarda invece il controllo che un template non debba essere anonimo

```
template = true => anonymous = false  :  template = false OR anonymous = false
```

Trigger

I trigger che verranno qui di seguito proposti sono stati ideati e realizzati per supportare i controlli che avvengono già a lato server. Infatti alcuni controlli avvengono già nel server e alcune funzionalità sono state implementate in modo che seguano determinate caratteristiche e convenzioni (ad esempio non può essere creato un form privo di domande). Talvolta abbiamo deciso di rafforzare questi controlli anche nel database utilizzando dei trigger e in alcuni casi delle funzioni che vengono richiamate con python.

Verifica che checkbox e radio button non presentino opzioni uguali

Trigger `differentCheckboxOption`:

```
create trigger "differentCheckboxOption"
before insert or update on form."CheckboxOption"
for each row
execute function form."differentCheckboxOption"()
```

Funzione `differentCheckboxOption()`:

```
CREATE OR REPLACE FUNCTION form."differentCheckboxOption"() RETURNS trigger LANGUAGE plpgsql AS $function$
begin
    if ( NEW.text in (select text from "form"."CheckboxOption" where id = NEW.id AND number <> NEW.number)) then
       return NULL;
    else
       return NEW;
    end if;
 end;
$function$;
```

Trigger `differentRadioOption`:

```
create trigger "differentRadioOption"
before insert or update on form."RadioOption"
for each row
execute function form."differentRadioOption"()`
```

Funzione `differentRadioOption()`:

```
CREATE OR REPLACE FUNCTION form."differentRadioOption"() RETURNS trigger LANGUAGE plpgsql AS $function$
begin
  if ( NEW.text in (select text from "form"."RadioOption" where id = NEW.id AND number <> NEW.number)) then
      return NULL;
  else
      return NEW;
  end if;
 end;
$function$;
```

I trigger appena mostrati abbiamo deciso di crearli per evitare che un utente possa creare domande checkbox o radio button contenenti opzioni tutte uguali. Questa scelta non mira a garantire l'integrità dei dati, bensì assume uno scopo funzionale ed estetico.

Verifica che quando si inserisce una risposta il form di riferimento sia attivo e non eliminato

Trigger `activeAndNotDeletedForm`:

```
create trigger "activeAndNotDeletedForm"
before insert or update on form."Answer"
for each row
execute function form."activeAndNotDeletedForm"()
```

Funzione `activeAndNotDeletedForm()`:

```
CREATE OR REPLACE FUNCTION form."activeAndNotDeletedForm"() RETURNS trigger LANGUAGE plpgsql AS $function$
begin
  if (true = (select active from "form"."Form" where id = NEW.form) and false = (select deleted from "form"."Form" where id = NEW.f
    return NEW;
  else
    RETURN NULL;
  end if;
end;
$function$;
```

Il trigger qui proposto verifica che quando un utente risponde ad un form, quest'ultimo sia attivo e non sia stato eliminato. In caso affermativo l'operazione di inserimento sarà portata a termine correttamente, altrimenti verrà abortita. Abbiamo deciso di creare questo form perché rispondere ad un form che è stato disattivato o è stato eliminato è logicamente scorretto e poiché l'unico controllo che avviene lato server è la query per ottenere il form e le relative domande dove si va a specificare che il form debba rispettare le condizioni `active = true` e `deleted = false`, quindi questo trigger rafforza i controlli.

Verifica che una sola radio option sia stata selezionata in una risposta

Trigger `oneRadioAnswer`

```
create trigger "oneRadioAnswer"
before insert or update on form."RadioAnswer"
for each row
execute function form."oneradioanswer"()
```

Funzione `oneradioanswer()`

```
CREATE OR REPLACE FUNCTION form."oneradioanswer"() RETURNS trigger LANGUAGE plpgsql AS $function$
  BEGIN
    if (new.id in (select id from "form"."RadioAnswer" where id = new.id and number <> new.number)) then
      raise exception 'In un radio button può essere scelta solo un opzione';
    else
      return new;
    end if;
  END;
$function$;
```

Questo trigger ci permette di controllare che in un radio button sia stata selezionata al più una opzione. Seppur anche HTML ci permette di controllare questo fenomeno grazie all'implementazione dei radio button appunto, abbiamo deciso di aggiungere questo ulteriore controllo, poiché se un utente dovesse trovare il modo di inserire più opzioni di una radio button allora ci sarebbe questo ulteriore controllo.

Funzioni

Oltre ai trigger abbiamo avuto la necessità di utilizzare delle funzioni che vengono chiamate esplicitamente. Questa necessita nasce dal fatto della decisione di utilizzare gli id delle varie entità come seriali e che vengono impostati automaticamente dal database. Gli scenari per i quali è necessaria questa pratica sono spiegati e motivati di seguito.

Verifica che a tutte le domande obbligatorie sia associata una risposta

Funzione `checkrequiredquestion(formid integer)`:

```
CREATE OR REPLACE FUNCTION form.checkrequiredquestion(formid integer)
  RETURNS boolean
  LANGUAGE plpgsql
 AS $function$
 declare
   test boolean;
   entry record;
 begin
```

```
    test = true;
    for entry in ( select id from "form"."Question" where required = true and form = formID ) loop
      if (entry.id not in (select q.question from "form"."CheckboxAnswer" q) and (entry.id not in (select q.question from "form"."Radio
        test = false;
      end if;
     end loop;
             return test;
end;
$function$;
```

La funzione qui proposta permette di verificare che quando vengono inserite tutte le risposte ad un form, se questo contiene delle domande obbligatorie, allora a queste deve essere associata una risposta. La necessità di usare una funzione e non un trigger deriva dal fatto che quando si inseriscono le risposte prima si inserisce l'entry di Answer e poi si inseriscono tutte le specifiche domande. Tuttavia in questo caso se si pone il trigger sulle varie tipologie di domande, allora dovrebbe essere attivato all'inserimento dell'ultima domanda, talvolta ciò non è possibile da identificare. Quindi dopo aver inserito tutte le risposte andiamo a richiamare esplicitamente la funzione per effettuare il controllo.

Verifica che un form non sia vuoto

Funzione `validform(formid integer)`

```
 CREATE OR REPLACE FUNCTION form.validform(formid integer)
   RETURNS boolean
   LANGUAGE plpgsql
  AS $function$
  BEGIN
       if ( (select count(*) from "form"."Question" where form = formID) < 1) then
             update "form"."Form"
             set deleted = true
             where id = formID;
             return false;
       else
             return true;
       end if;
       END;
  $function$;
```

Questa funzione ci permette di verificare se un form che è appena stato creato è valido, ovvero se contiene almeno una domanda. Come per la funzione precedente non è chiaro identificare l'ultima domanda che è stata inserita nel form utilizzando un trigger, quindi effettuiamo l'inserimento di tutte le domande (se ce ne sono) e poi invochiamo esplicitamente questa funzione per controllare che il form sia valido. Questo controllo risulta un rafforzamento poiché la creazione del form e delle domande realizzato in HTML e JavaScript non permettono la realizzazione di un questionario vuoto, talvolta il gruppo ha deciso che per aumentare il livello di sicurezza per il quale non si verifichi il caso di un form privo di domande fosse una buona pratica aggiungere questo controllo sotto forma di funzione.

Indici

Oltre agli indici che vengono creati automaticamente dal database sugli attributi che sono chiavi primarie abbiamo ritenuto opportuni creare altri 2 indici in modo da migliorare le performance delle operazioni di lettura e modifica sulle tabelle interessate.

I due indici che abbiamo creato sono

Indice Question

```
CREATE INDEX "questionForm" ON "Question" USING btree (id, form)
```

Abbiamo trovato utile aggiungere questo indice perché quando si vogliono recuperare le domande appartenenti ad un form, se queste risultano essere ordinate secondo il form di appartenenza allora le operazioni risulterebbero più efficienti.

Indice Form

```
CREATE INDEX "makerForm" ON "Form" USING btree (id, maker)
```

Un altro indice che abbiamo trovato potesse essere utile è quello qui indicato, che ordina le righe della tabella Form secondo l'id e il maker, questo perché quando un utente vuole visualizzare la lista dei suoi Form allora le operazioni di lettura

risulterebbero più efficienti dal momento che nel database le righe di Form che hanno lo stesso attributo maker sono contigue.

Altre scelte progettuali

Condivisione del form per rispondere

Abbiamo deciso di utilizzare la stessa tecnica di Google Form per la condivisione dei form, trovandola per il lato utente la più facile ed immediata nonostante sia un sito che necessita di una registrazione, questa soluzione ha anche però dei difetti, cioè un ipotetico utente può indovinare l'id di un questionario a lui non destinato, inserendo così la sua risposta, tuttavia abbiamo ritenuto comunque più valida questa opzione per facilitare e migliorare l'esperienza utente. Questo problema potrebbe comunque essere ovviato cifrando il link o l'id che viene passato per rispondere, rendendo così impossibile scrivere un id o un link in chiaro che però non è stato ricevuto dal proprietario.

Ruolo

La decisione da noi presa è quella di non avere più di un ruolo questo per la semplice natura del sito in generale, tutti gli utenti secondo noi dovevano avere la possibilità di creare un form e di risponderci come è per Google Form, l'unico ruolo sensato da poter inserire è quello di 'admin' che permette di vedere tutti i form, questo non è stato inserito semplicemente per una scelta riguardante la privacy.

Qui di seguito viene mostrata la creazione del ruolo, e i privilegi che esso può esercitare sulla nostra base di dati:

```
CREATE ROLE "user" WITH
  NOSUPERUSER
  NOCREATEDB
  NOCREATEROLE
  NOINHERIT
  LOGIN
  NOREPLICATIONGRANT INSERT,UPDATE,SELECT ON Users
  NOBYPASSRLS;
GRANT INSERT,UPDATE,SELECT,REFERENCES,TRIGGER ON Answer;
GRANT INSERT,UPDATE,SELECT,REFERENCES,TRIGGER ON CheckboxAnswer;
GRANT INSERT,UPDATE,SELECT,REFERENCES,TRIGGER ON CheckboxOption;
GRANT INSERT,UPDATE,SELECT,REFERENCES,TRIGGER ON CheckboxQuestion;
GRANT INSERT,UPDATE,SELECT,REFERENCES,TRIGGER ON FileAnswer;
GRANT INSERT,UPDATE,SELECT,REFERENCES,TRIGGER ON FileQuestion;
GRANT INSERT,UPDATE,SELECT,REFERENCES,TRIGGER ON Form;
GRANT INSERT,UPDATE,SELECT,REFERENCES,TRIGGER ON OpenAnswer;
GRANT INSERT,UPDATE,SELECT,REFERENCES,TRIGGER ON OpenQuestion;
GRANT INSERT,UPDATE,SELECT,REFERENCES,TRIGGER ON Question;
GRANT INSERT,UPDATE,SELECT,REFERENCES,TRIGGER ON RadioAnswer;
GRANT INSERT,UPDATE,SELECT ON RadioOption;
GRANT INSERT,UPDATE,SELECT,REFERENCES,TRIGGER ON RadioQuestion;
GRANT INSERT,UPDATE,SELECT,REFERENCES,TRIGGER ON User;
```

File CSV

Abbiamo deciso di dare la possibilità all'utente di scaricare i propri form anche in formato csv, dove questo file contiene tutte le domande e per ognuna di queste tutte le risposte. Per realizzare questa funzionalità abbiamo usato la libreria csv che semplifica notevolmente il lavoro. In particolare ci creiamo una lista che contiene tutti i testi delle domande o per quanto riguarda le risposte i testi di queste, le opzioni selezionate o il nome del file, in base al tipo di domanda. Successivamente andiamo a creare il file all'interno del server, questo viene creato in scrittura poiché viene creato, riempito e poi chiuso e mai più riaperto. Infine grazie alla libreria csv andiamo a scrivere i nostri dati, usando la funzione `writewor()` che come parametro accetta una lista e quindi andrà a stampare ogni elemento della lista come una cella della stessa riga.

Download file delle risposte

Per quanto riguarda l'upload di file come risposta ad una domanda abbiamo appunto memorizzato i file, dopo averli ricevuti e ottenuti dal pacchetto HTTP, all'interno del server. Il proprietario del form che può quindi anche visualizzare le risposte ottenute può effettuare il download del file caricato. Per implementare questa funzionalità ci siamo serviti della funzione `send_file()` dove come parametro viene specificato il file da inviare nel caso in cui la route venga richiamata dall'utente, quindi quest'ultimo scaricherà il file richiesto.

Ulteriori informazioni

Per poter sviluppare questo progetto abbiamo utilizzato Git per gestire il versioning del codice in questo modo ognuno di noi poteva lavorare su porzioni di codice diverse senza intralciare il lavoro l'uno con l'altro. Una problematica è stata quella di non avere un database unico per tutti infatti ognuno di noi ne aveva uno locale e ad ogni modifica il trasferimento di dump tra di noi è stato prolisso e poco efficiente soprattutto perché non tutti utilizzavano lo stesso database tool (DBeaver o phpPgAdmin).

Le view dell'applicazione sono state realizzate con HTML, basandosi sul framework Bootstrap per permettere la responsiveness e per utilizzare un'estetica quanto più piacevole.

Una parte di codice è stata scritta in linguaggio JavaScript per permettere la dinamicità nella creazione di form/template e delle relative domande, rendendo il processo più gratificante esteticamente e molto più intuitivo per l'utente. Il codice è stato diviso in 3 parti dove il contenuto appartiene allo stesso contesto e alla stessa logica, in particolare abbiamo il primo file, `general.js`, che contiene funzioni di carattere generico, poi abbiamo `form.js` e `template.js`, i quali invece sono file che contengono le funzioni per creare e modificare rispettivamente le domande dei form e dei template.

Altro utilizzo di JavaScript è stato per utilizzare le DataTable, tabelle che sfruttano librerie CSS e JavaScript per creare tabelle interattive, in grado di ordinare le righe secondo un attributo, in senso crescente o decrescente, di visualizzare più o meno righe di una tabella lasciando la scelta all'utente e navigare fra le varie pagine della tabella nel caso in cui ci sia un numero di righe maggiore di quello massimo visualizzabile.

Una scelta interessante è quella del funzionamento dei template, infatti l'idea di inserire dei template e di permetterne la creazione ci è venuta ancora una volta prendendo ispirazione da Google Form, tuttavia l'implementazione è diversa, infatti nel nostro progetto sarà possibile creare template e modificarli, ma una volta importati per la creazione del form le domande del template non saranno modificabili questo per una nostra scelta infatti era per noi insensato soprattutto perché avrebbe reso l'opzione template obsoleta e di poca importanza.

Un altro punto già citato sopra cioè del poter rispondere ad un form una sola volta è stata per noi una scelta decisa soprattutto dopo l'analisi attenta e dettagliata di un programma che permetta a tutti gli utenti di accedere con facilità ad un form, ma con necessità di un'iscrizione all'applicazione web. Utilizzando questo ultimo punto si poteva garantire una sorta di sicurezza in più limitando la risposta ad una per utente evitando anche se in modo marginale la "falsificazione" dei questionari.

Infine come ultima, ma non meno importante, della funzionalità è quella della visualizzazione dei form. Sempre ispirandoci a Google Form abbiamo deciso di permettere all'utente di visualizzare le risposte in due modalità: modalità riepilogo dove per ogni domanda del form può osservare tutte le relative risposte ottenute, e modalità individui, dove le risposte, per ogni domanda, si suddividono per ogni utente. In quest'ultimo caso la visualizzazione varia nel caso in cui il form sia anonimo o meno: nel primo caso il proprietario del form non sarà in grado di verificare a chi appartengono le risposte, nel secondo caso sarà in grado di vedere la mail dell'utente che ha risposto.