Web App curricula - Ingegneria Informatica Unifi

Francesco Ermini

November 21, 2018

1 Requirements

Le funzioni che devono essere supportate lato amministratore sono:

- Caricamento dati relativi a ciascun insegnamento: codice (ob), nome (ob), CFU (ob), SSD (ob), URL scheda insegnamento, periodo di erogazione, anno di programmazione
- Caricamento dei possibili percorsi con nome del percorso e descrizione
- Specifica della struttura di ciascun percorso: gruppi di esami a scelta vincolata
- Modifica e cancellazione di insegnamenti, percorsi e loro struttura

Le funzioni che devono essere supportate lato studente sono:

- Immissione dati studente (matricola, nome e cognome)
- Visualizzazione dei possibili percorsi e loro descrizione
- Scelta del percorso
- Visualizzazione dell'elenco dei possibili esami per il percorso scelto con accesso al link della scheda insegnamento per ciascun esame
- Scelta, per ciascun gruppo a scelta vincolata previsto dal percorso, degli insegnamenti all'interno del gruppo
- Visualizzazione del piano di studi specificato
- Creazione di PDF e file di uscita a descrivere il piano scelto

Tecnologie di realizzazione:

- PostgreSQL,
- Python
- Flask

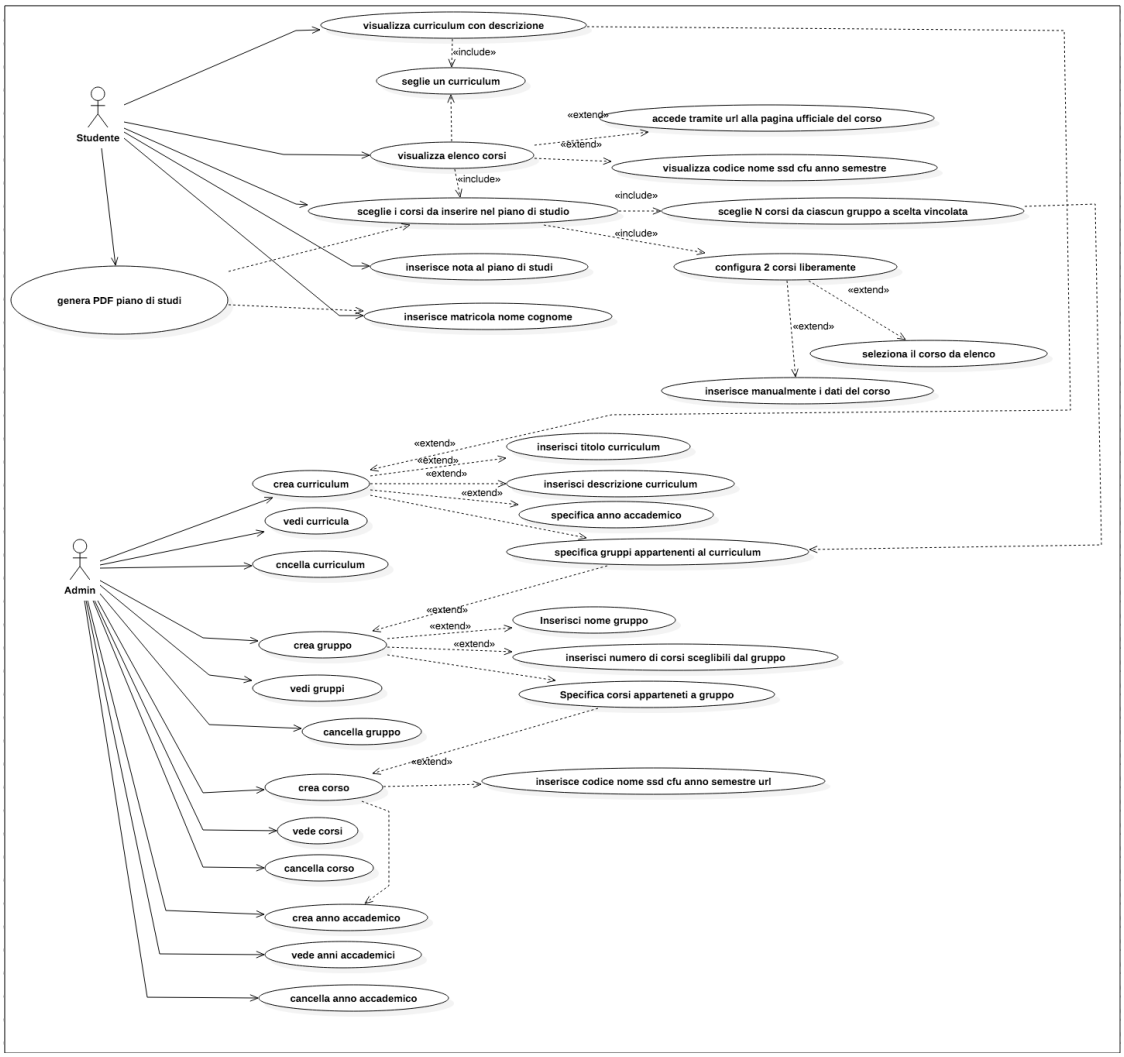

Figure 1: Use case diagram - simplified version

2 Data model

Dai requirements si sono ricavate le seguenti entità:

- Student
- Studyplan
- Curriculum
- Course
- Group
- Academicyear

Student consiste in una matricola (identificativo univoco di *Student*), un nome e un cognome.
Studyplan consiste in un curriculum ed un insieme non nullo di corsi scelti nel rispetto dei vincoli specificati dall'amministratore. L' *id* è uguale alla matricola dello studente a cui appartiene.
Curriculum consiste in un titolo, una descrizione, un anno accademico e un insieme non nullo di gruppi.
Group consiste in un nome, un numero n (numero di corsi da scegliere per quel gruppo), un valore di *cfu* e un insieme non nullo di *Courses*. Rappresenta una tabella di esami a scelta vincolata per un certo *Curriculum*.
Course consiste in un codice corso (identificativo univoco del corso), un nome, un settore disciplinare, un numero di cfu, un anno accademico, l'anno ed il semestre in cui è previsto lo svolgimento del corso ed un url alla pagina ufficiale del corso.
Academicyear consiste in una stringa che rappresenta l'anno accademico (i.e 2018-2019), la data di inizio anno accademico e la data di fine anno accademico.

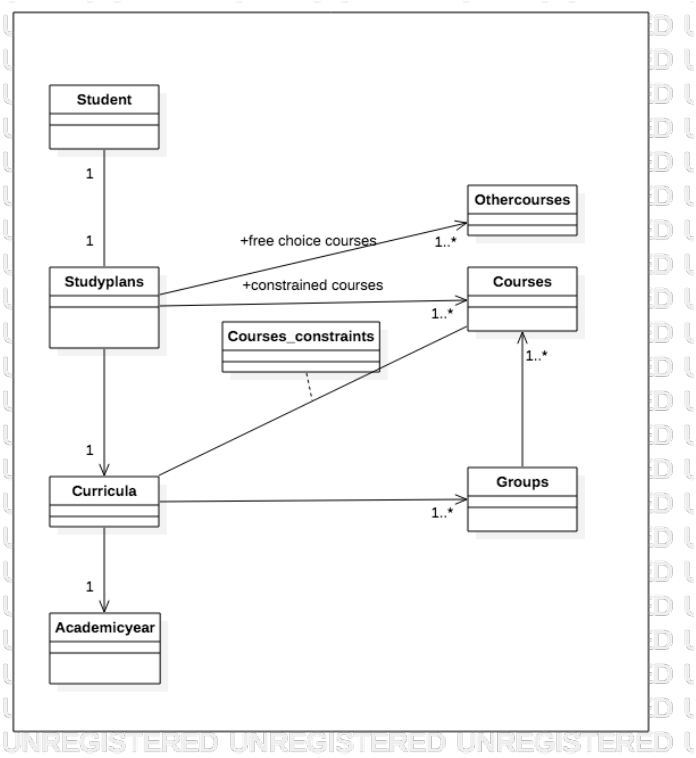

Figure 2: UML concettuale

Dai casi d'uso in 1 sono stati ricavate le relazioni mostrate in figura 2

Table 1: Entity relations

ER	**cardinality**
Student - Studyplans	one-to-one bidirectional
Studyplans - Curricula	many-to-one bidirectional[1]
Studyplans - Courses	many-to-many unidirectional
Curricula - Groups	many-to-many unidirectional
Groups - Courses	many-to-many unidirectional

In particolare si evidenziano le osservazioni:

- **Student - Studyplan**: Ad uno studente deve corrispondere uno e un solo piano di studi. Ad un piano di studi deve corrispondere uno e un solo studente. la relazione è stata modellata in modo bidirezionale, così che da uno studente si possa ricavare il suo piano di studi e viceversa. Se cancello uno studente cancello il suo piano di studi e se cancello un piano di studi cancello lo studente a cui appartiene.
- **Studyplan - Curricula** : Ad un piano di studio deve corrispondere uno e uno solo curriculum. Un curriculum è in uso su zero o molti piani di studio. Sebbene dai casi d'uso non sia emersa

la necessità di sapere i piani di studio associati ad un curriculum, questa relazione è stata fatta bidirezionale per consentire l'eliminazione di un curriculum; Per cancellare un curriculum e non violare i vincoli di integrità (foreign key su *Studyplan*) è infatti necessario avere una relazione tra curriculum e piani di studio. Si sottolinea che così facendo l'eliminazione di un curriculum renda nullo il campo *curriculum_ id* in tutti i piani di studio che lo avevano scelto. Questa opzione è stata considerata accettabile in fase di progettazione. D'altra parte l'amministratore deve avere totale controllo sulla eliminazione dei curricula. Inoltre se l'amministratore decide di eliminare un curriculum, i piani di studio che lo avevano scelto non saranno più validi.

- **Studyplan - Courses**: Ad un piano di studi devono corrispondere uno o più corsi. Un corso sarà incluso in zero o molti piani di studio. La relazione è stata fatta unidirezionale perché non esiste un caso d'uso che richiede di sapere i piani di studio associati ad un corso. Quando l'amministratore elimina un corso, quel corso è automaticamente rimosso dai piani di studio che lo avevano scelto.

- **Curricula - Groups**: Un curriculum è fatto di uno o più gruppi. Un gruppo può essere incluso in zero o molti curricula. La relazione è stata fatta unidirezionale perché non esiste un caso d'uso che richieda di sapere i curricula associati ad un gruppo. L'eliminazione di un gruppo provoca la rimozione di quel gruppo dai curricula.

- **Groups - Courses** : Un gruppo è fatto da uno o più corsi. Gruppi diversi possono avere alcuni corsi in comune (a patto che questi gruppi non appartengano ad uno stesso curriculum). Anche in questo caso non esiste un caso d'uso che richieda di sapere, dato un corso, quali gruppi appartengono a quel corso.

La figura 3 illustra il diagramma ER in prospettiva di implementazione.
Il database è stato realizzato in `Postgres` tramite `SQLAlchemy` in `Python Flask`.

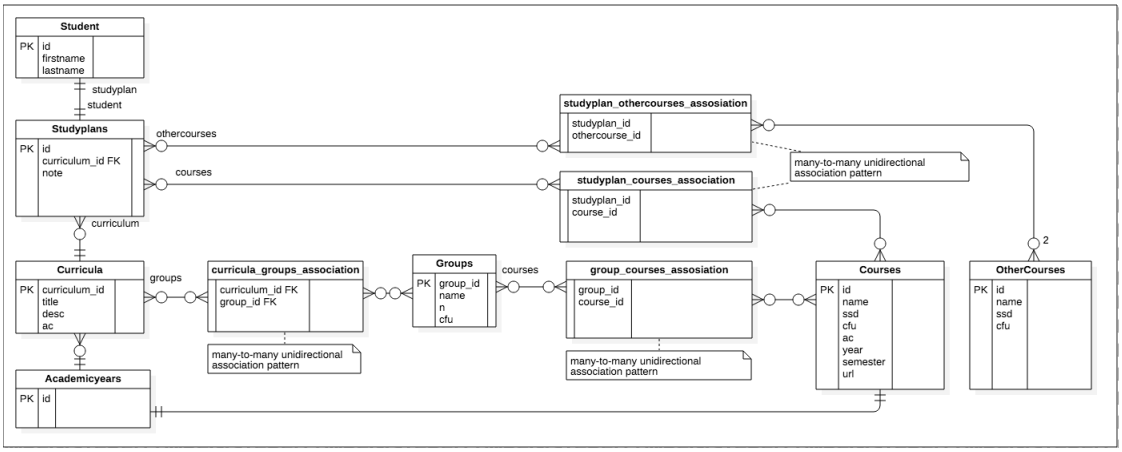

Figure 3: ER diagram 2

3 Project Setup and structure

Prima di procedere al setup assicurarsi di avere Python e pip installati. (io ho python *2.7.10* e pip *18.0*)

Listing 1 Setup Flask

```
mkdir curriculaWebApp && cd curriculaWebApp
pip install virtualenv
virtualenv env
source env/bin/activate
git init
git clone https://github.com/FrancescoErmini/CurriculaWebApp2.git
export FLASK_APP=app.py
pip install Flask
pip install -r requirements.txt
```

Poi dopo aver scaricato Postgres sul proprio sistema operativo, accedere alla console *psql*, quindi creare un database, creare un utente e associare utente al database.

Listing 2 Create database

```
CREATE USER pala WITH PASSWORD 'password';
CREATE DATABASE pianodistudio2;
GRANT ALL PRIVILEGES ON DATABASE pianodistudio2 TO pala;
```

in alternativa, scaricare **pgAdmin 4** e fare tutto graficamente.
Prima di procedere oltre verificare che nel file *app.py* i dati inseriti sopra siano corretti:

Listing 3 Flask db configuration

```
16 POSTGRES = {
17     'user': 'pala',
18     'pw': 'password'
19     'db': 'pianodistudio2',
20     'host': 'localhost',
21     'port': '5432',
22 }
23 app.config['DEBUG'] = True
24 app.config['SQLALCHEMY_DATABASE_URI'] = 'postgresql://%(user)s:\
25 %(pw)s@%(host)s:%(port)s/%(db)s' % POSTGRES
```

A questo punto si procede alla migrazione del database seguendo lo schema di migrazione [**?**].

Listing 4 Initialize database

```
python manage.py db init
python manage.py db migrate
python manage.py db upgrade
python manage.py runserver
```

Una volta avviato il server si può consumare le API sia da riga di comando che da interfaccia grafica.

4 Validazione

4.1 validazione e vincoli sui dati

La validazione del contenuto dei dati è utile per verificare i dati inviati tramite JSON post. In particolare *Academicyear, Course, Curriculum, Group* serviranno all'utente *admin* per limitare l'inserimento di dati incorretti. Invece la validazione delle entità *Student e Studyplan* serviranno allo studente per controllare il rispetto dei vincoli.

4.1.1 Academicyear

Nel caso di *Academicyear* l'id è del tipo `2018-2019`. La validazione consiste nel:

1. controllare che il valore sia non nullo
2. controllare che il carattere '-' separi gli anni.
3. controllare che l'anno indicato si trovi in un range di valori ragionevoli, nel caso tra 2018 e 2022
4. controllare che l'anno successivo al primo sia quello successivo.

```
academicyear = data['id']
years = string.split(academicyear, '—')
if academicyear == "":
  return jsonify({"error":"Academic year can not be empty"}),500
if len(years) < 2:
  return jsonify({"error":"Wrong format. use minus char — to separate years"}),500
if int(years[0]) < 2018 or int(years[0]) > 2021:
  return jsonify({"error": "Year must be in range [2018,2021] "}),500
if (int(years[0]) + 1) != int(years[1]):
  return jsonify({"error": "Year not in format [<year>—<year+1>]" }),500
```

4.1.2 Course

Nel caso di *Course* l'id è del tipo `B012345`. La validazione consiste nel

1. controllare che il valore sia non nullo
2. controllare che il codice abbia esattamente 7 caratteri

```
course_id = data['id']
if course_id == "":
  return jsonify({"error":"Course code can not be empty"}),500
if len(course_id) != 7:
  return jsonify({"error":"Course code is 7 char length"}),500
```

4.1.3 Student

Nel caso di *Student* l'id è del tipo `1234567`. La validazione consiste nel

1. controllare che il valore sia non nullo
2. controllare che il codice abbia esattamente 7 caratteri

```
student_id = data['id']
if student_id == "":
  return jsonify({"error":"Student id can not be empty"}),500
if len(student_id) != 7:
  return jsonify({"error":"Student id is 7 char length"}),500
```

4.1.4 Studyplan

Nel caso di *Studyplan* non è necessario controllare l'id in quanto vincolato all'id dello *Student*. La validazione consiste nel:

1. verificare che non ci siano corsi duplicati (unicità del corso).
2. verificare che per ogni gruppo del curriculum siano stati scelti almeno n corsi.

```
 uniqueCourses = []
        for course in data['courses']:
          if course['id'] not in uniqueCourses:
            uniqueCourses.append(course['id'])
          else:
            return  jsonify({"error": "duplicate courses"}),500

        for group in curriculum.groups:
          count=0
          for course in data['courses']:
            if( course['id'] in [c.id for c in group.courses]):
              count+=1
          if count < group.n:
            return jsonify({"error": "group " + str(group.name) +" has " +str(count)+"
                 courses but " + str(group.n) + " courses are expected for curriculum
                " +  str(curriculum.title) }),500
```

4.1.5 Group

1. controllare che ci sia almeno un corso
2. controllare che il vincolo di corsi da scegliere non sia zero
3. controllare che il numeri di corsi presenti nel gruppo sia superiore al minimo numero di corsi che lo studente può scegliere da quel gruppo.

4. controllare che i corsi di un gruppo siano unici

```
group = request.get_json()

    if len(group['courses']) < 1:
      return jsonify({"error":"No courses specified. Create course first, then specify
           group courses."}), 500

    if group['n'] < 1:
      return jsonify({"error":"N must be a positive number"}), 500

    if len(group['courses']) <= group['n']:
      return jsonify({"error": "group has " + str(len(group['courses'])) + " courses
          but " + str(group['n']) + " courses are expected."}), 500

    uniqueCourses = []
    for course in group['courses']:
      if course['id'] not in uniqueCourses:
        uniqueCourses.append(course['id'])
      else:
        return jsonify({"error":"duplicate course in group"}),500
```

4.1.6 Curriculum

1. controllare che il titolo sia non nullo
2. controllare che ci siano almeno 1 gruppi.
3. controllare che non ci siano corsi uguali in gruppi diversi, ovvero che i corsi di un curriculum siano unici
4. (non implementato) controllare che corsi e cfu specificati dal gruppo siano coerenti

```
curriculum  = request.get_json()

    if curriculum['title'] == "":
      return jsonify({"error": "title can not be empty"}), 500

    if len(curriculum['groups']) == 0 :
      return jsonify({"error": "No groups. specified. Create groups first, then
          specify curriculum groups."}), 500


    uniqueCourses = []
    for g in curriculum['groups']:
      group = Groups.query.get(g['id'])
      for course in group.courses:
```

```
14          if course.id not in uniqueCourses:
15            uniqueCourses.append(course.id)
16          else:
17            return jsonify({"error": "duplicate courses in curriculum" + str(len(
                  uniqueCourses))}),500
```

5 Vincoli su creazione e cancellazione

- La creazione delle entità deve avvenire in modo ordinato per rispettare le dipendenze.
- La cancellazione delle entità non deve essere vincolata alle dipendenze con gli elementi della base dati.

In particolare si evidenzia che l'ordine di creazione delle entità deve rispettare il seguente ordine:

1. Creazione dell'anno accademico
2. Creazione dei corsi
3. Creazione dei gruppi
4. Creazione dei curricula

Per quanto riguarda la cancellazione delle entità, il sistema è stato progettato per permettere all'amministratore la cancellazione di tutte le entità della base di dati. L'inconsistenza dei vincoli espressi nelle relazioni è verificata a livello applicativo. Di seguito sono analizzate le relazioni. In particolare si evidenzia che:

5.1 Cancellazione di un corso

La cancellazione di un corso provoca la rimozione del corso nei gruppi e nei piani di studi che lo contengono. La cancellazione di un corso da un piano di studi non provoca la cancellazione del piano di studi. La validità di un piano di studi è verificata nel momento in cui viene creato. Se successivamente alla sua creazione uno dei corsi presenti nel piano di studi venisse rimosso, tale invalidità verrebbe notificata allo studente con un messaggi di errore La cancellazione di un corso da un gruppo non garantisce l'invalidità di quel gruppo e non provoca la sua cancellazione. Il caso in cui la rimozione del corso provoca l'invalidità del gruppo si considera al pari della rimozione del gruppo stesso.

```
1  groups_courses_association = db.Table('groups_courses_association', Base.metadata,
2      db.Column('group_id', db.Integer, db.ForeignKey('groups.id')),
3      db.Column('course_id', db.String(7), db.ForeignKey('courses.id',  ondelete='
           cascade'), nullable=True))
4
5  studyplan_courses_association = db.Table('studyplan_courses_association', Base.
       metadata,
6      db.Column('studyplan_id', db.String(7), db.ForeignKey('studyplans.id')),
7      db.Column('course_id', db.String(7), db.ForeignKey('courses.id',  ondelete='
           cascade'), nullable=True))
```

5.2 cancellazione di un gruppo

La cancellazione di un gruppo provoca la rimozione di quel gruppo da un curriculum ma non la cancellazione del curriculum. La validità del curriculum è determinata dal numero di cfu accumulati dalla scelta di n corsi da ciascuno dei gruppi presenti. Per questo motivo la cancellazione di un gruppo potrebbe causare l'invalidità di un curriculum.

```
curricula_groups_association = db.Table('curricula_groups_association', Base.metadata,
    db.Column('curriculum_id', db.Integer, db.ForeignKey('curricula.id')),
    db.Column('group_id',  db.Integer, db.ForeignKey('groups.id')))
```

5.3 cancellazione di un curriculum

La cancellazione di un curriculum provoca la cancellazione dei piani di studio che lo contengono.

La scelta di cancellare i piani di studi riferiti a curriculum non più esistenti serve a mantenere pulito il database, dando modo all'amministratore di rimuovere i piani di studi obsoleti

```
curriculum = db.relationship('Curricula', backref=db.backref('studyplans', uselist=
    False,  cascade='all,delete'))
```

6 Page navigation

Table 2: Page navigation

directory	descrizione
/	vedi elenco dei curricula e scegli curriculum
/curriculum/1	vedi corsi disponibili per curriculum, seleziona corsi e inserisci dati per piano di studi
/studyplan/1234567	vedi piano di studi e stampa pdf
/admin/index	vedi pannello di controllo
/admin/passwd	cambia credenziali di accesso dell'amministratore
/admin/course/index	Vedi lista dei corsi disponibili e accedi alle opzioni CRUD per corsi
/admin/course/create	Inserisci dati per creazione di un nuovo corso
/admin/course/read/1	Leggi dettagli corso
/admin/course/delete/1	Cancella corso
/admin/group/index	Vedi lista dei gruppi disponibili e accedi alle opzioni CRUD per gruppi
/admin/group/create	Inserisci dati per creazione di un gruppo
/admin/group/read/1	Leggi dettagli gruppo
/admin/group/delete/1	Cancella gruppo
/admin/curriculum/index	Vedi lista dei curriculum disponibili e accedi alle opzioni CRUD per curriculum
/admin/curriculum/create	Inserisci dati per creazione di un nuovo
/admin/curriculum/read/1	Leggi dettagli curriculum
/admin/curriculum/delete/1	Cancella curriculum

7 CSV Parser

A partire dal file csv fornito si è implementato uno script python che effettua il parse dei dati del csv. Una volta isolati i dati d'interesse per un corso, lo script assembla e invia in automatico una richiesta POST per creare un nuovo corso.

Listing 5 Data parser

```
csvfile = open("course2.csv", "r")
csvfile.readline()
for line in csvfile:
  row = line.split(";")
```

Lo script esclude dal parser i due casi di 'non corso': Uno riguarda la prova finale e l'altro il laboratoriotirocinio. Oltre che per una questione logica, i due devono essere evitati perché non avendo il campo 'semestre' mandano in errore il parser. Inoltre le 'string' analizzate post-parser includono l'aggiunta di uno spazio vuoto a fine della parola. Questo spazio è eseguito con:

Listing 6 String manupulation

```
    id = row[3]
    id = id[:-1]
    ssd = row[5]
    ssd = ssd[:-1]
    name = row[4]
    name = name[:-1]
```

Infine la richiesta viene forgiata con:

Listing 7 send JSON POST req

```
payload = {
  "id": id,
  "name": name,
  "cfu": int(row[6]),
  "ac": "2018-2019",
  "year": int(row[0]),
  "semester": int(row[1]),
  "ssd": ssd,
  "url": "url"
};
url = 'http://localhost:5000/course/'
headers = {'Content-Type': 'application/json'}
data = json.dumps(payload)
resp = requests.post(url, data=data, auth=HTTPBasicAuth('admin', 'admin'), headers=
    headers)
```

References

[1] Flask migration https://medium.com/@dushan14/create-a-web-application-with-python-flask-postgresq

[2] Many-to-Many in SQLAlchemy http://docs.sqlalchemy.org/en/latest/orm/basic_relationships.html#many-to-many

[3] Flask Tutorial https://danidee10.github.io/2016/09/19/flask-by-example-2.html

[4] Base class schema in SQLAlchemy https://medium.com/@lsussan/base-classes-one-to-many-relationships-in-flask-sqlalchemy-fba0d47374ad