Web App curricula - Ingegneria Informatica Unifi

Francesco Ermini

November 28, 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

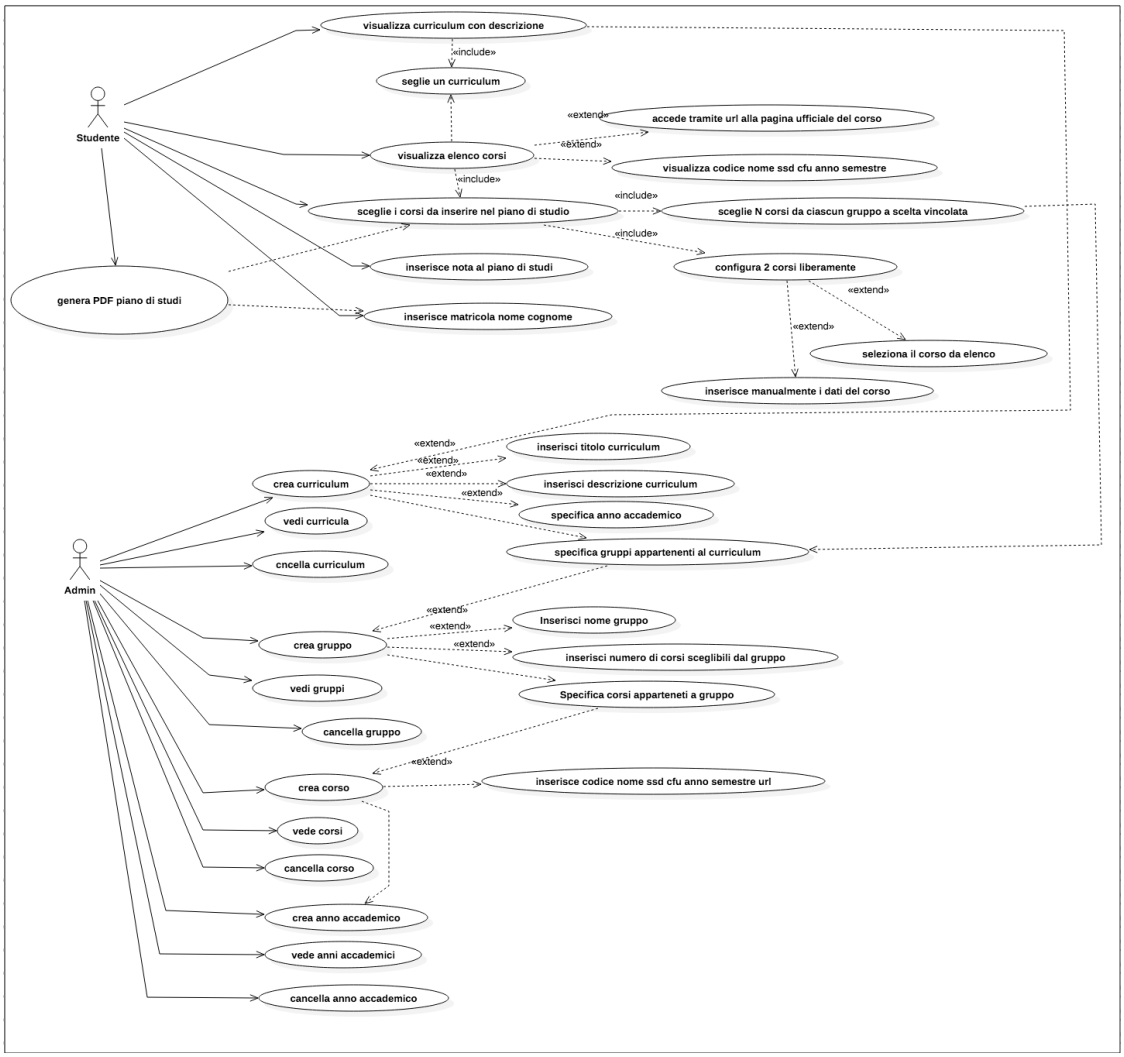

Figure 1: Use case diagram - simplified version

2 Data model

Dai requirements si sono ricavate le seguenti entità:

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

Student modella uno studente. Lo studente è identificato da una matricola. Matricola è *id* di *Student*. Inoltre è caratterizzato da un nome e un cognome, *firstname* e *lastname*.
Studyplan modella un piano di studi. Un piano di studi è identificato dallo studente a cui appartiene. L'*id* di *Studyplan* è uguale a *id* di *Student*. Un piano di studi è caratterizzato da uno e un solo *Curriculum*, da un certo numero di *Courses* a scelta vincolata e da due corsi a scelta libera, *Othercourses*. Inoltre un piano di studi può contenere una *note* testuale.
Curriculum modella un curriculum. E' identificato da un *id*, numero automaticamente dato dal sistema. Un *Curriculum* è caratterizzato un titolo, una descrizione e un anno accademico, rispettivamente *title, desc, ac*. Inoltre un curriculum è caratterizzato da un insieme di gruppi. vedi *Group*.
Group è un raggruppamento di corsi scelti dall'amministratore secondo un criterio di affinità degli stessi. Un gruppo serve a definire i criteri di selezione dei corsi a scelta. vincolata. Ciascun gruppo è caratterizzato da un *name* con funzione descrittiva, per esempio 'gruppo Automazione 01'. A ciascun gruppo è associato un insieme di *Courses*. In analogia a quanto avviene nei piani di studio cartacei, un *Group* rappresenta una tabella di corsi a scelta vincolata all'interno del quale lo studente deve scegliere obbligatoriamente *n* corsi tra quelli contenuti nel gruppo. *Group* è identificato da un *id*, numero automaticamente dato dal sistema.
Course Rappresenta un corso a scelta vincolata appartenente all'insieme dei corsi disponibili per la scuola a cui è rivolta l'applicazione, in questo caso la scuola di ingegneria informatica dell'università di Firenze. *Course* è creato dall'amministratore del sistema ed inserito nel relativo database. Un *Course* è identificato da un codice 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, rispettivamente *id, name, ssd, cfu, ac, year, semester, url*.
Othercourse Rappresenta un corso a scelta libera che può non appartenere ai corsi disponibili per la scuola considerata, da cui il nome *Othercourse*. I corsi presenti in *Othercourse* sono creati dallo studente ed inseriti nel relativo database. Questi corsi hanno un numero di attributi minore rispetto a *Course*. In particolare sono composti da un codice corso, un nome, un settore disciplinare, un numero di cfu, rispettivamente *id, name, ssd, cfu*.
Academicyear consiste in una stringa che rappresenta l'anno accademico (i.e 2018-2019) ed è anche identificativa per l'entità.

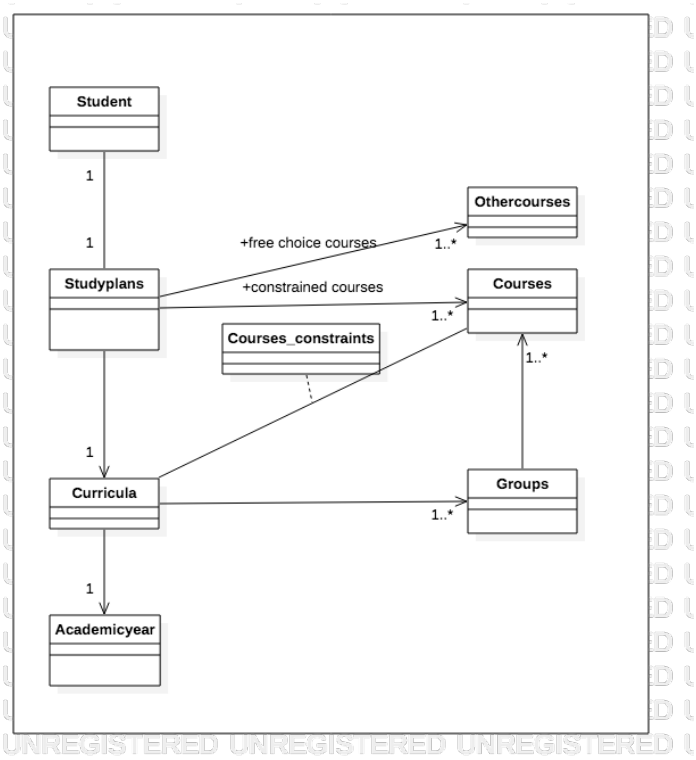

Figure 2: UML concettuale

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
Studyplan - Othercourses	many-to-many unidirectional [2]
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 a scelta vincolata. 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.

- **Studyplan - Courses**: Ad un piano di studi devono corrispondere due corsi a scelta libera.

- **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`.

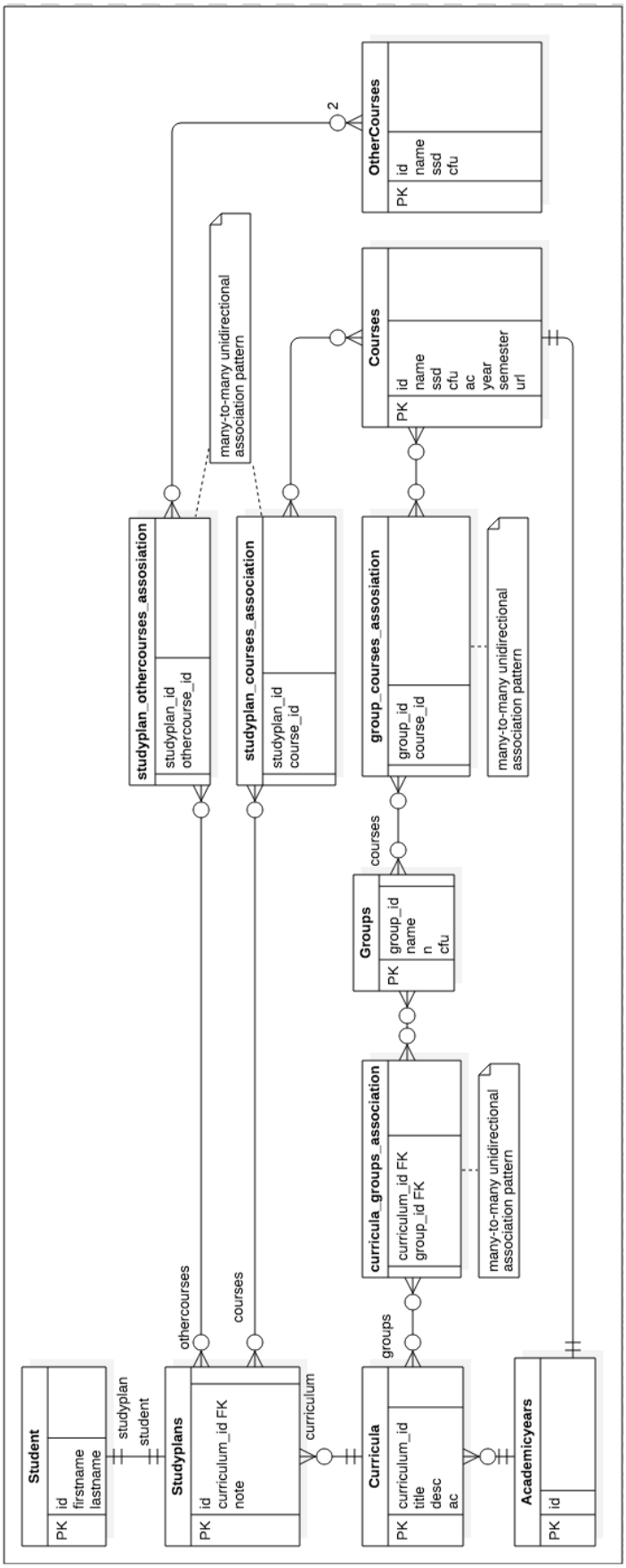

Figure 3: ER diagram

3 Creazione e validazione delle entità

- **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.

- **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
  3. controllare che il nome del corso non sia vuoto
  4. controllare che il numero dei cfu sia o 6 o 9
  5. controllare che il campo ssd non sia vuoto

- **Group**

  1. controllare che il nome del gruppo non sia vuoto
  2. controllare che il gruppo non sia privo di corsi
  3. controllare che il numero di corsi da scegliere nel gruppo non sia zero
  4. controllare che il numeri di corsi presenti nel gruppo sia superiore al minimo numero di corsi che lo studente può scegliere da quel gruppo.

  Nota: L'unicità dei corsi in un gruppo è verificata all'interno di curriculum.

- **Curriculum**

  1. controllare che il titolo sia non nullo
  2. controllare che il curriculum non sia privo di gruppi
  3. controllare che non ci siano corsi uguali per id in gruppi diversi, ovvero che i corsi di un curriculum siano unici
  4. controllare che corsi e cfu specificati dal gruppo siano sufficienti a totalizzare 84 cfu (numero di crediti dei corsi a scelta vincolata)

  Nota: Il controllo sulla unicità del corso è stato inserito in considerazione del fatto che dalle tabelle riportate nei requirements non si è visto nessun corso duplicato in più gruppi all'interno di un curriculum.

- **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

- **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.

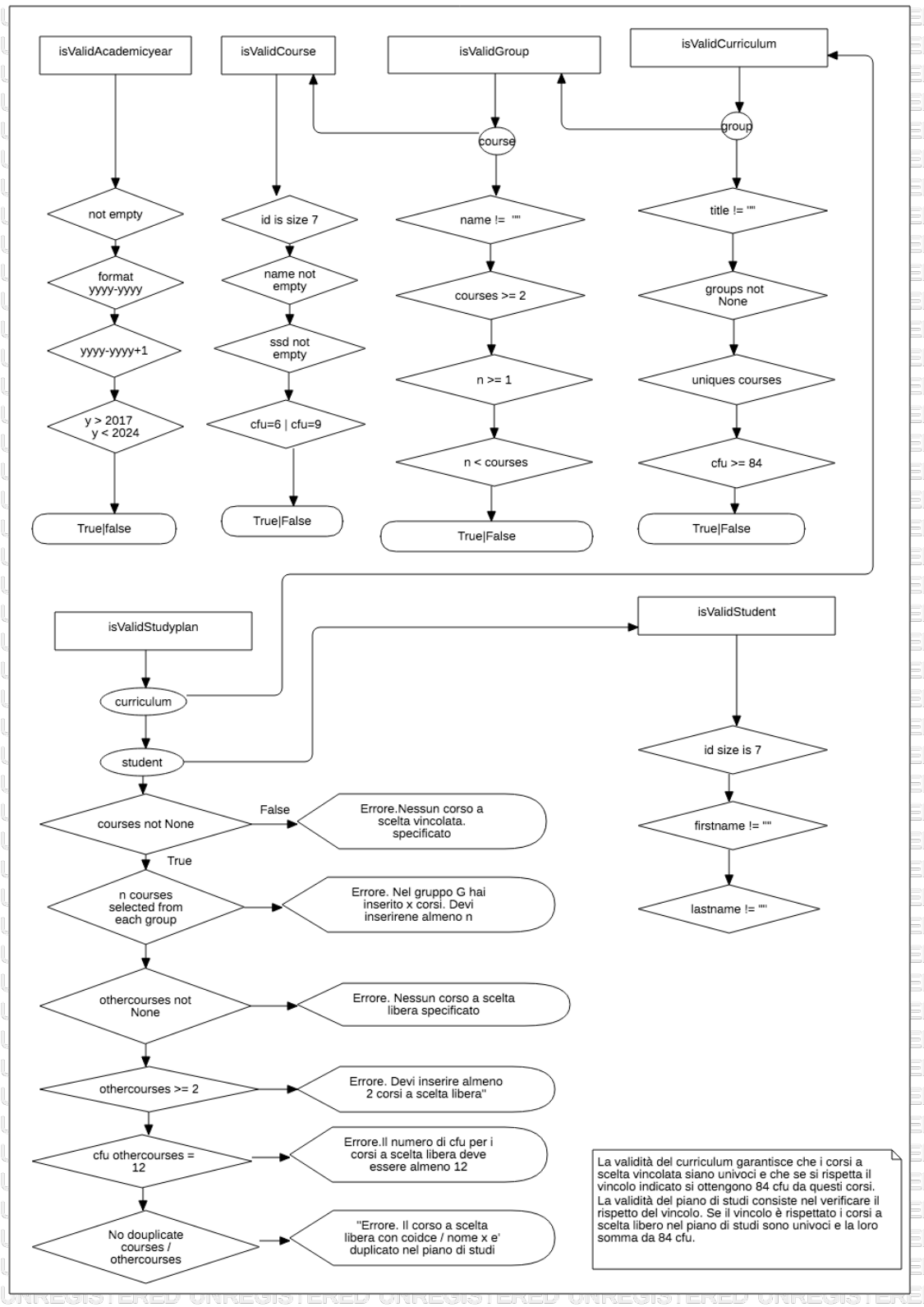

Figure 4: Validazione - Flow diagram

3.1 Creazione e cancellazione di entità e relazioni in SQLAlchemy

La creazione di una entità deve rispettare l'ordine di creazione delle entità da cui dipende. Il motivo per cui non si è potuto fare altrimenti è che la validazione di una entità dipende per lo più dai dati delle altre entità. Ad esempio, la validazione del curriculum richiede che siano noti i corsi che fanno parte di quel curriculum. Per evitare la possibilità di creare entità non valide si è preferito costringere l'amministratore ad eseguire la creazione ordinata delle entità. In particolare si evidenzia che l'ordine di creazione delle entità deve rispettare il seguente ordine:

1. Creazione dell'anno accademico *Academicyears*
2. Creazione dei corsi *Courses*
3. Creazione dei gruppi *Groups*
4. Creazione dei curricula *Curricula*
5. creazione di *Othercourses*
6. creazione di *Student*
7. creazione di *Studyplans*

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:

- **curricula-groups** Many-to-Many unidirectional association pattern con padre Curricula e figlio Groups. La cancellazione di un gruppo da parte dell'amministratore provoca la rimozione di quel gruppo dai curricula che lo contenevano ma non la cancellazione del curricula. Tuttavia può darsi che la cancellazione di un gruppo da un curriculum invalidi il curriculum a causa del non raggiungimento degli 84 cfu necessari.

```
1  class Curricula(Base):
2      __tablename__ = 'curricula'
3    ...
4      groups = db.relationship('Groups', secondary='curricula_groups_association')
5
6  curricula_groups_association = db.Table('curricula_groups_association', Base.
       metadata,
7      db.Column('curriculum_id', db.Integer, db.ForeignKey('curricula.id')),
8      db.Column('group_id',  db.Integer, db.ForeignKey('groups.id',  ondelete='
           cascade')))
```

- **groups-courses** Many-to-Many unidirectional association pattern con padre Groups e figlio Courses. La cancellazione di un corso provoca la rimozione di quel corso dai gruppi che lo contenevano. Occorre verificare che il gruppo sia ancora valido. In particolare che il numero di corsi presenti nel gruppo sia superiore al numero di corsi obbligatori da scegliere.

```
class Groups(Base):
    __tablename__ = 'groups'
  ...
    courses = db.relationship('Courses', secondary='groups_courses_association')

groups_courses_association = db.Table('groups_courses_association', Base.metadata
    ,
    db.Column('group_id', db.Integer, db.ForeignKey('groups.id')),
    db.Column('course_id', db.String(7), db.ForeignKey('courses.id',  ondelete='
        cascade')))
```

- **studyplan-courses** Many-to-Many unidirectional association pattern con padre Studyplan e figlio Courses. La cancellazione di un corso da un piano di studi invalida il piano di studi ma non lo cancella. Si assume che lo scopo dell'applicazione è quello di favorire lo studente nella creazione del PDF relativo al piano di studi. Lo studente che accede all'applicazione in un dato momento configura il proprio piano di studi con i corsi presenti nel database in quel momento. Se, a distanza di un anno dal primo piano di studi creato, lo studente intende rinnovare il piano di studi per l'anno successivo dovrà compilarlo ex-novo con i corsi disponibili nel sistema.

```
class Studyplans(Base):
    __tablename__ = 'studyplans'
  ..
    courses = db.relationship('Courses', secondary='studyplan_courses_association
        ')

studyplan_courses_association = db.Table('studyplan_courses_association', Base.
    metadata,
    db.Column('studyplan_id', db.String(7), db.ForeignKey('studyplans.id')),
    db.Column('course_id', db.String(7), db.ForeignKey('courses.id',  ondelete='
        cascade')))
```

- **studyplan-othercourses** Many-to-Many unidirectional association pattern con padre Studyplan e figlio Othercourses Nel sistema non è stato implementato nessun meccanismo di cancellazione dei corsi dalla tabella othercourses.

```
class Studyplans(Base):
    __tablename__ = 'studyplans'
  ..
    othercourses = db.relationship('OtherCourses', secondary='
        studyplan_othercourses_association')

studyplan_othercourses_association = db.Table('studyplan_othercourses_association
    ', Base.metadata,
    db.Column('studyplan_id', db.String(7), db.ForeignKey('studyplans.id')),
    db.Column('othercourse_id', db.String(7), db.ForeignKey('othercourses.id',
        ondelete='cascade')))
```

- **studyplan-curriculum** Many-to-one bidiectional pattern. In questo caso la bidirezionalità, in particolare da curriculum verso studyplan, è stata aggiunta per permettere all'amministratore di cancellare i piani di studio che rimangono senza curriculum a seguito della cancellazione dello stesso. 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.

```
class Studyplans(Base):
    __tablename__ = 'studyplans'
  ..
    curriculum = db.relationship('Curricula', backref=db.backref('studyplans',
        uselist=False,  cascade='all,delete'))
```

- **Studyplan-Student** One-to-One bidirectional pattern con padre Student e figlio Studyplan Studente e piano di studi sono fortemente accoppiati. Se cancello il piano di studio, cancello anche lo studente che lo possiede. L'inserimento di ondelete='Cascade' all'interno della ForeignKey è stato aggiunto per conferire robustezza al codice; non esiste un meccanismo che permette la cancellazione diretta dello studente. Solo in seguito a cancellazione di un curriculum, studenti e piani di studio possono essere cancellati dall'amministratore.

```
class Studyplans(Base):
    __tablename__ = 'studyplans'
    id = db.Column(db.String(7), db.ForeignKey('students.id', ondelete='CASCADE')
        , primary_key=True)
    student = db.relationship('Students', backref=db.backref('studyplans',
        uselist=False), cascade='delete')
```

4 Page navigation diagram

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 s
/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
/admin/studyplan/index	Vedi tutti i piani di studio
/admin/studyplan/read/5972786	vedi dettagli del piano di studi

5 Autenticazione admin

L'autenticazione è stata implementata tramite Flask simple login[5]. Ad ogni operazione di creazione o cancellazione di entità è stato aggiunto il decoratore

```
@login_required
```

Tramite questo decoratore quando si prova ad accedere all'url protetto da login si viene reindirizzati ad una pagina di login in cui inserire username e password. Successivamente l'applicazione manterrà valida l'autenticazione per la durata della sessione.
La prima volta che viene fatta partire l'applicazione username e password sono inizializzati ad 'admin - admin'.

```
def init_user_password():
    if  Admin.query.get(0) is None:
        username = 'admin'
        password = generate_password_hash('admin', method='pbkdf2:sha256')
        admin = Admin(idd=0, username=username, password=password)
```

Successivamente tramite interfaccia grafica alla pagina /admin/passwd sarà possibile modificare username e password con valori a propria scelta. Si noti che:

- La password non è memorizzata in chiaro. Viene memorizzato il digest dell'hash.
- La tabella Admin ammette un solo utente (con id=0)

6 CSV Parser

A partire dal file csv fornito si è implementato uno script python che effettua il parse dei dati del csv e crea automaticamente i corsi.

Listing 1 Data parser

```
csvfile = open("csv/corsi.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. Leliminazione dello spazio è eseguito con:

Listing 2 String manupulation

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

L'anno accademico è inserito da admin da interfaccia grafica. L'url che non è disponibile è momentaneamente settato con link a pagina interna.

```
                url = "#"
                ac = request.form['academicyear']
```

7 Generazione del pdf relativo al piano di studi

La generazione del pdf lato server richiede l'installazione dell'applicazione

```
sudo apt-get install wkhtmltopdf
```

La creazione del pdf e l'invio dello stesso al client è gestita da:

```
  studyplanDoc = render_template('studyplan.html', studyplan=studyplan, student=
      student)
    pdf = pdfkit.from_string(studyplanDoc, False)
    response = make_response(pdf)
    response.headers['Content-Type'] = 'application/pdf'
    response.headers['Content-Disposition'] = 'inline; filename=pianodistudi.pdf'
```

8 Guida all'installazione dell'applicazione

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

Listing 3 Setup Flask

```
mkdir curriculaWebApp && cd curriculaWebApp
pip install virtualenv
virtualenv env
source env/bin/activate
git init
git clone https://github.com/FrancescoErmini/CurriculaWebApp4.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 4 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 5 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 6 Initialize database

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

In aggiunta occorre installare su Debian/Ubuntu una applicazione che si occupa della creazione del pdf lato server:

```
sudo apt-get install wkhtmltopdf
```

8.1 Deploy dell'applicazione su Server Ubuntu

Per le istruzione sul deploy si rimanda alla guida: https://medium.com/ymedialabs-innovation/deploy-flask-app-with-nginx-using-gunicorn-and-supervisor-d7a93aa07c18

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

[5] Flask simple login https://github.com/rochacbruno/flask_simplelogin