[Flask-SQLAlchemy]

General Information & Licensing

Code Repository	https://github.com/pallets/flask-sqlalchemy/
License Type	BSD
License Description	● A permissive license similar to the BSD 2-Clause License, but with a 3rd clause that prohibits others from using the name of the project or its contributors to promote derived products without written consent.
License Restrictions	● Liability ● Warranty
Who worked with this?	Jason Liu

[Object SQLAlchemy]

Purpose

This object allows us to save any important information related to the users and the multimedia uploaded. With this object, we instantiate it and can create and alter a database using SELECT/INSERT/DELETE methods. Specifically, this allows us to manage information on user's usernames, nicknames, passwords, profile preferences and image upvote count.

Location of code: n/a not implemented

Magic

This object needs to be instantiated before using SQLAlchemy to create and alter databases. We can instantiate the object by using:

```
db = SQLAlchemy(app)
```

which then provides a class called Model which is a declarative base which can be used to declare models. The Model class is used to describe a database where if we had multiple databases, each database would have their own properties.

To initialize SQLAlchemy, Flask app must be created and added the config:

```
['SQLALCHEMY_DATABASE_URI'] = <name of db>
```

which then needs to be called using the method db.create_all(). This method takes the db we initialized and creates all the tables. From here, we are able to take the classes we created using the Model class to INSERT into the table:

```
// Model class used to describe the User class
class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(80), unique=True, nullable=False)
    password= db.Column(db.String(120), unique=True, nullable=False)

// Here we can INSERT into the db using the User class we created
user1 = User(username='Adam', password='weakpassword123')
user2 = User(username='Ben', email='weakpassword321')
```

Location of code: n/a not implemented

[Class Model]

Purpose

The Model class allows us to create multiple tables for our database. Such tables include a Users table and Image table. They will create and store columns appropriate to each table. For the User’s table, the username, nickname, password, and UID will be the columns. For the Image’s table, the upvote count, downvote count, and UID will be the columns.

Location of code: n/a not implemented

Magic

Creating the model was discussed in the object SQLAlchemy section. To use it we can create as many classes (tables) as we want like this (for User and Image):

```
class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(80), unique=True, nullable=False)
    nickname = db.Column(db.String(80), unique=True, nullable=False)
    password = db.Column(db.String(120), unique=True, nullable=False)

class Image(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    upvote = db.Column(db.Integer, unique=False, nullable=False)
    downvote = db.Column(db.Integer, unique=False, nullable=False)
```

After creating these classes, we can now INSERT to each table a new user or image object:

```
user1 = User(username='Adam', nickname='Ad', password='weakpassword123')
image1 = Image(upvote=1, downvote=0)
```

Location of code: n/a not implemented

[Method add (INSERT)]

Purpose

This method is used to insert new data into our db. The inserted data needs to be of a db.Model class. Raw data cannot be inserted into tables.

Location of code: n/a not implemented

Magic

To add data to the db, we first create a db.Model object using one of the classes we created before (User or Image class with db.Model). As mentioned, raw data cannot be added to the tables so we must instantiate a new Model:

```
user1 = User(username='Adam', nickname='Ad',password='weakpassword123')
```

This is then used to add into the existing db:

```
db.session.add(user1)
db.session.commit()
```

As simple as that, user1 is now in the db. This can also be done with other classes such as Image:

```
image1 = Image(upvote=1, downvote=0)
db.session.add(image1)
db.session.commit()
```

Must remember to commit the changes using db.session.commit()

Location of code: n/a not implemented

[Method delete (DELETE)]

Purpose

This method is used to delete data from our db. The data to be deleted needs to be in the db or the delete method will not work.

Location of code: n/a not implemented

Magic ⋆★˚｡⋆୨୧˚ ☽ ˚｡⋆

To delete data from the db, we need the id from the entry in the database. This can be obtained by calling the added object with .id or selected from the database:

```
user1 = User(username='Adam', password='weakpassword123')
item_to_delete = user1.id // This gives us the id for user1

// Similar to the add method, we can call the delete method
db.session.delete(item_to_delete) // Notice we have the user1.id as the argument to delete
db.session.commit()
```

As simple as that, user1 is now deleted from the db. This can also be done with other classes such as Image:

```
image1 = Image(upvote=1, downvote=0)
db.session.delete(image1.id)
db.session.commit()
```

Must remember to commit the changes using db.session.commit()

Location of code: n/a not implemented

[Method query (SELECT)]

Purpose

This method is used to select data from our db. The data to be selected is determined by using a filter. If no data with the filter can be found, the method will return None. The query method can return a single item or multiple items.

Location of code: n/a not implemented

Magic

To select data from the database, we need to filter out what we want to be returned. For instance say we want user1 from the db. There are multiple ways we can filter user1 from the db:

```
select1= User.query.filter_by(username='Adam').first()
select2 = User.query.filter_by(nickname='Ad').first()
```

Since we have both username and nickname columns, we have 2 ways of returning user1. If we want to return the first user we can also filter by id:

```
select3 = User.query.filter_by(id=1).first()
```

If the filter does not find any data to select, it will return None instead.
If we want to select multiple users:

```
select_multiple = User.query.filter(User.nickname.startswith('A')).all()
```

would return all users that has a nickname that starts with the letter ‘A’.

Location of code: n/a not implemented