Portfolio Project – FishingStories

Author: Robert J McGinness

Question 1:

This project utilizes object relational mapping for database interaction. Rather than use flask-sqlalchemy, pure sqlalchemy is used here. The reason for avoiding flask-sqlalchemy is to provide a web framework-independent interface to the database. This, of course, could have been achieved with a raw SQL implementation. However, this project provides the opportunity to utilize ORM to gain greater experience, there is a short development time associated with the course duration, and it allows for programmatic migration. To implement the latter, the flask-migrate package is applied. In order to allow cooperation of the pure sqlalchemy and flask-migrate modules, an adapter method was used.

Question 2:

Endpoints:

Fishingstories:

GET ‘/index’, ‘/’

auth:

POST ‘/auth’

POST ‘/auth/register’

logout:

GET ‘/logout’

admin:

GET ‘/admin’

GET ‘/admin/anglers’

GET ‘/admin/ anglers/{angler\_id}’

PUT ‘/admin/anglers/{angler\_id}/edit’

GET ‘/admin/account\_types

GET ‘/admin/account\_types/{account\_type\_id}’

POST ‘/admin/account\_types/create’

PUT ‘/admin/account\_types/{ account\_type\_id}/edit’

GET ‘/admin/privileges’

GET ‘/admin/privileges/{privilege\_id}’

POST ‘/admin/privileges/create’

PUT ‘/admin/privilees/{privilege\_id}/edit’

DELETE ‘/admin/privileges/{ privilege\_id}’

GET ‘/admin/users’

GET ‘/admin/users/{user\_id}’

PUT ‘/admin/users/{user\_id}/edit’

GET’/admin/users/{user\_id}/fish’

GET ‘/admin/baits’

GET ‘/admin/gear’

GET ‘/admin/fishing\_spots’

GET ‘/admin/fishing\_outings’

GET ‘/admin/fish’

GET ‘/admin/fish/{fish\_id}’

GET ‘/admin/fish/{fish\_id}/fishing\_spot/{spot\_id}’

GET ‘/admin/fish/{fish\_id}/anglers’

GET ‘/admin/ranks’

GET ‘/admin/ranks/{rank\_id}’

POST ‘/admin/ranks/create’

PUT ‘/admin/ranks/{rank\_id}/edit’

GET ‘/admin/baits’

GET ‘/admin/gear’

GET ‘/admin/stats’

user:

GET ‘users/{user\_id}’

PUT ‘users/{user\_id}/edit’

GET ‘/users/{user\_id}/baits’

GET ‘/users/{user\_id}/baits/{bait\_id}’

POST ‘/users/{user\_id}/baits/create’

PUT ‘/users/{user\_id}/baits/{bait\_id}/edit’

GET ‘/users/{user\_id}/gear’

GET ‘/users/{user\_id}/gear/{gear\_id}’

POST ‘/users/{user\_id}/gear/create’

PUT ‘/users/{user\_id}/gear/{gear\_id}/edit’

GET ‘/users/{user\_id}/fish’

GET ‘/users/{user\_id}/fish/{fish\_id}’

POST ‘/users/{user\_id}/fish/create’

DELETE ‘/users/{user\_id}/fish/{fish\_id}’

GET ‘/users/{user\_id}/fishing\_spots’

GET ‘/users/{user\_id}/fishing\_spots/{spot\_id}’

POST ‘/users/{user\_id}/fishing\_spots/create’

PUT ‘users/{user\_id}/fishing\_spots/{spot\_id}/edit’

GET ‘users/{user\_id}/fishing\_spots/{spot\_id}/fish’

GET ‘users/{user\_id}/fishing\_spots/stats’

GET ‘users/{user\_id}/fishing\_spots/{spot\_id}/stats’

GET ‘/users/{user\_id}/fishing\_outings’

GET ‘/users/{user\_id}/fishing\_outings/{outing\_id}’

POST ‘/users/{user\_id}/fishing\_outings/create’

PUT ‘users/{user\_id}/fishing\_outings/{outing\_id}/edit’

Question 3:

Future versions, could implement clustering of anglers into groups. Then endpoints can be adjusted to aggregate data by the group. For example, fishing\_spots known by the group, fish caught by the group, statistics for the group. This would require database design changes. With the current database design, there could be tournaments created. This would require searching over a date/time frame for fish caught by anglers. Creating a tournament would be an admin endpoint. Getting tournament results or entering a tournament would be endpoints for the user.

Question 4:

The flask-sqlalchemy package does make ORM easier. In particular, it manages sessions and database connectivity for you. Anytime you abstract a concept or process, it is likely that access to functionality is limited. The design decision to use pure sqlalchemy required managing access to the database. This is done through creation of a scoped session, as threading in flask limits the possibilities for accessing the database. Assigning the session to the app allows access to it in other modules through the context of the flask current\_app.

As described above, this design decision also required the creation of an adaptor mechanism to allow utilization of sqlalchemy in flask-migrate. In the end, it is an elegant solution, utilizing duck typing on an object that mimics a SQLAlchemy object, but maintains the context for the database.

One additional matter, that has not been overcome as of now is the creation of an initial admin user\_account. Several possible mechanisms include a super user app that connects to the fishing\_stories database or running a separate Python script to insert the record. Registration of a new user intentionally does not allow creation of a user\_account with Administrator privileges. However, creation of a new user\_account internally, unknown to the user, creates an angler record associated with that new user. It also requires pre-created account\_types, privileges, and ranks. Creation of these records is only allowed to the administrator. Thus the Administrator privilege has to exist prior to an Administrator user\_account is created. The same user\_accounts table stores the users and Administrators. Additionally, it is not reasonable to run a SQL script directly to create the Administrator user\_account, as password hashing is performed via Python. This necessitates use of a separate meta\_user app or a separate Python script to create the Administrator record. A SQL script could be run directly on the database to create privilege records.