CREATING AN API

Create a blueprint in our app. Works from our database whereby the use connects to an API route, we query our database for relevant information, organize that information into json data and send back to the user as json data

Things to consider when creating an API.

1. What data are we providing?
2. Do we have an existing data set?
3. What is the main format of our data?
4. What functionality do we want to interact with this data?

Our Answers For Class Exercise

1. What data are we providing DB Model- designing a model for the animals in an exotic pet shop
2. Do we have an existing data set? No, we’ll make some stuff by creating a database
3. What is the main format of our data? Main format- Database Model-design this first

.DB Model-designing a model for the animals in an exotic pet shop

-Id uuid= string

-Name = string

-Scientific Name= string

-size= string (needs to specify units)

-weight= integer

-diet= string

habitat = string

-lifespan = interger

-realease date= Numeric

-description = string

-price = numeric(float w/2 decimals)

A.In the models file start laying out the new database model, the columns that are going to go into this model.

B.Now we can think about how to structure the init method. Since we are going to be allowing users to input new things into the api the init has to be a dictionary the init for this is going to be different than the other table for the user because we

#are going to let some users create new data in our database and create a new animal(the user will be sending

# the 'POST' request in Json formatt to our system which translates into a python dictionary). Therefore,

#our init method for creating a new animal has to accept a single dictionary.

C. Now we can update my database to have this model

#1activate my virtual environment

#2update database flask db migrate

#3 flask db upgrade

#test init method by adding some data by giving flask shell access to my database model,over to run.py

#import the Animal model next to the user model and in the context processor

#in terminal type flask shell (enter), then db(enter): to access my db, then the name of the model:i.e User, Animal, Movies whatever your table name is (enter)

# to create a test of animals and the scenario of required info and unrequired info need to create a dictionary since that is the format of my init

# type in terminal ad = {}, then add key value pairs to it by saying ad['name']='Fennec Fox'(enter), ad['sci\_name']='Vulpes zerda'

#now type ad to print the dict.(enter), then type fox = Animal(ad)"ad is the name of the dictionary (enter), then type fox(enter), then fox.\_\_dict\_\_(enter)

#check everything is right then type db.session.add(fox), db.session.commit(), then check if it worked by querying the database. type Animal.query.all() or go check elephant database

D. Now we can start creating our API. Over to number 4

1. What functionality do we want to interact with this data?

Create a new api blueprint inside of the app folder if there is not one already and call the new folder api. Make another file inside of the api folder and call it routes.py

A.)

#intial blueprint setup

from flask import Blueprint

#instantiate blueprint api and then create a connection between the blueprint api and my flask app.

api = Blueprint('api', \_\_name\_\_, url\_prefix='/api')

B.) #Head over to init.py

C.) head over to the init.py under import blueprints type: from .api.routes import api

Then under register blueprints type:app.register\_blueprint(api)

D.) #head over to routes to make your routes.

E.) #the decorator belonging to a blueprint starts with @<blueprint\_name> rather than @app

#below is a test route

@api.route('/test')

def test():

return jsonify({'database': 'whoa this is some cool data'})

#jsonify? transorms python dictionary (or list) into json data, must import jsonify next to blueprints to use

reaad all data from database

#the decorator belonging to a blueprint starts with @<blueprint\_name> rather than @app

#below is a test route

@api.route('/test')

def test():

#jsonify? transorms python dictionary (or list) into json data, must import jsonify next to blueprints to use

return jsonify({'database': 'whoa this is some cool data'})

#make sure you run code often so do a flask run and go to website to see if it is working

#API ROUTES FOR MY ANIMALS MODEL

#route for getting all animals-start with this 1st (easier)-

@api.route('/animals', methods=['GET'])

def getAnimals():

"""

[Get] return json data on all of the animals in our database

"""

#query the animals

#I want to jsonify the result of .to\_dict() for each animal in our animals query

animals = [a.to\_dict() for a in Animal.query.all()]

#jsonify and send

return jsonify(animals)

-read single data from database

#route for getting one animal-this is going to be a dynamic route/

# we don't want to create a route for every single animal we want to get.that would be too many routes!

# this route will expect input coming from through the url

@api.route('/animal/name/<string:name>', methods=['GET'])

def getAnimal(name):

"""

[GET] That accepts an animal name through the url and either gets

the appropriate animal from our database or returns that we dont have that animal

"""

a = Animal.query.filter\_by(name=name.title()).first()# we use the .title so when the user types in the search by name it will capaitalize and return whether they use lowercase name or uppercase name

if a:

return jsonify(a.to\_dict()), 200

else:

return jsonify({'Request failed': 'No animal with that name.'}), 404

-create new data

the init for this is going to be different than the other table for the user because we are going to let some users create new data in our database and create a new animal(the user will be sending the 'POST' request in Json formatt to our system which translates into a python dictionary). Therefore, our init method for creating a new animal has to accept a single dictionary.

#route for creating new animal-we are expecting to get some information. We expect the user input information to match the structure we created with the dictionary. To access the body of a post request we need to do this through the flask request object.

1. in the api routes form up top next to jsonify type: request
2. in the api routes scroll up to imports and write db to import the database next to from app.models import Animals

now it should say from app.models import db, Animal

#route for creating new animal-we are expecting to get some information

@api.route('/create/animal', methods=['POST'])

def create\_animal():

"""

[POST] creates a new animal in our database with data provided in the request body

expected data format

"""

#how do we accept request in the body of a post request

# depending on how specific we want our data to be - we may want to build out some checks

# on the data coming in, does it actually make sense? is it something we want in our database?

#otherwise, create the new animal in the database

try:

data = request.get\_json() #grab any json data from the body of the request made to this route

new\_animal = Animal(data)

db.session.add(new\_animal)

db.session.commit()

return jsonify({'Created New Animal': new\_animal.to\_dict()})

except:

return jsonify({'Create Animal Rejected': 'Animal already exists or improper request.'})

#we cant test this create animals route because ['GET'] method doesnt exist for the url

#create animal only the ['POST']. to test we would have to write the request in python or use an API

#testing tool

1. head over to a jupyter notebook to check the python way(reference the jupyter notebook you made)
2. postman is an api testing tool that you can use alternatively to test your api post.head to the website account you created.

Tip:add some status codes by these routes. Reference the vs code document api routes

-updata data

#route for updating an animal

@api.route('/animal/update/<string:id>', methods=['PUT'])#put is used for updating existing data-just like POST, PUT requests can include data being sent to the web server

def updateAnimal(id):

"""

[PUT] accepts an animal ID in the URL and JSON data in the PUT request body in the following format(all values optional)

{

'name': <str>,

'sci\_name: <str>,

'description': <str>,

'price': <float>,

'image': <str>

###rest of k:v pairs optional

'size': <str>,

'weight': <int>,

'diet': <str>,

'habitat': <str>,

'lifespan': <int>

}

"""

try:

#grab the request body and query the database for an animal with that ID

animal = Animal.query.get(id)

data = request.get\_json()

#then update animal object.The dictionary coming in will only have coming in what we

# want to change. Have to make a change to our model. We need to make a revers of our

#to\_dict model called a from dict method

animal.from\_dict(data)#rewrite yourself from this dictionary function we created

#and recommit it to the database aka it already exists in the database so in this case it is just saveing the changes

db.session.commit()

return jsonify({'Updated animal': animal.to\_dict()}), 200

except:

return jsonify({'Request failed': 'invalid body or animal ID'}), 400

-delete data

#route for deleting an animal

@api.route('/animal/remove/<string:id>', methods=['DELETE'])

def removeAnimal(id):

#if the animal is present in the database say so, and remove

animal = Animal.query.get(id)

if not animal: #if no animal with that id is in the database

#tell the user remove failed

return jsonify({'Remove failed':'No animal of that ID exists in the database'}), 404

db.session.delete(animal)

db.session.commit()

return jsonify({'Removed animal': animal.to\_dict()}), 200

Answers for the homework assignment- on my own practice

1. What data are we providing? DB Model- designing a model for the horror movies available to buy from shop of horrors video store.
2. Do we have an existing data set? No make some stuff up
3. What is the main format of our data? in format- Database Model-design this first

-Id uuid= string

-Name = string

-Rating= string

-Box office= integer

-Director= string

-category = string

-price = numeric(float w/2 decimals)

A.In the models file start laying out the new database model, the columns that are going to go into this model.

B.Now we can think about how to structure the init method. Since we are going to be allowing users to input new things into the api the init has to be a dictionary the init for this is going to be different than the other table for the user because we

#are going to let some users create new data in our database and create a new animal(the user will be sending

# the 'POST' request in Json formatt to our system which translates into a python dictionary). Therefore,

#our init method for creating a new animal has to accept a single dictionary.

C. Now we can update my database to have this model

#1activate my virtual environment

#2update database flask db migrate

#3 flask db upgrade

#test init method by adding some data by giving flask shell access to my database model,over to run.py

#import the Animal model next to the user model and in the context processor

#in terminal type flask shell (enter), then db(enter): to access my db, then the name of the model:i.e user, animal, whatever your table name is (enter)

# to create a test of animals and the scenario of required info and unrequired info need to create a dictionary since that is the format of my init

# type in terminal ad = {}, then add key value pairs to it by saying ad['name']='Fennec Fox'(enter), ad['sci\_name']='Vulpes zerda'

#now type ad to print the dict.(enter), then type fox = Animal(ad)"ad is the name of the dictionary (enter), then type fox(enter), then fox.\_\_dict\_\_(enter)

#check everything is right then type db.session.add(fox), db.session.commit(), then check if it worked by querying the database. type Animal.query.all() or go check elephant database

D. Now we can start creating our API. Over to number 4

1. What functionality do we want to interact with this data?

Create a new api blueprint inside of the app folder if there is not one already and call the new folder api. Make another file inside of the api folder and call it routes.py

A.)

#intial blueprint setup

from flask import Blueprint

#instantiate blueprint api and then create a connection between the blueprint api and my flask app.

api = Blueprint('api', \_\_name\_\_, url\_prefix='/api')

B.) #Head over to init.py

C.) head over to the init.py under import blueprints type: from .api.routes import api

Then under register blueprints type:app.register\_blueprint(api)

D.) #head over to routes to make your routes.

E.) #the decorator belonging to a blueprint starts with @<blueprint\_name> rather than @app

#below is a test route

@api.route('/test')

def test():

return jsonify({'database': 'whoa this is some cool data'})

#jsonify? transorms python dictionary (or list) into json data, must import jsonify next to blueprints to use

-read all data from database

#API ROUTES FOR MY MOVIES MODEL

#route for getting all movies

@api.route('/movies', methods=['GET'])

def getMovies():

"""

[Get] return json data on all of the animals in our database

"""

#query the movies

#I want to jsonify the result of .to\_dict() for each movie in our movies query

movies = [m.to\_dict() for m in Movies.query.all()]

#jsonify and send

return jsonify(movies)

This is a get method that specifying is optional but we should so we can keep track. We are just sending information to the user.

Need to import our Animal model into the routes folder

@api.route('/animals', methods='GET')

-read single data from database

the init for this is going to be different than the other table for the user because we are going to let some users create new data in our database and create a new animal(the user will be sending the 'POST' request in Json formatt to our system which translates into a python dictionary). Therefore, our init method for creating a new animal has to accept a single dictionary.

#route for creating new animal-we are expecting to get some information. We expect the user input information to match the structure we created with the dictionary. To access the body of a post request we need to do this through the flask request object.

A.) in the api routes form up top next to jsonify type: request

B.) in the api routes scroll up to imports and write db to import the database next to from app.models import Animals

now it should say from app.models import db, Animal

#route for creating a new movie-we are expecting to get some information

@api.route('/create/movie', methods=['POST'])

"""

[POST] creates a new movie in our database with data provided in the request body

expected data format

"""

#how do we accept request in the body of a post request

# depending on how specific we want our data to be - we may want to build out some checks

# on the data coming in, does it actually make sense? is it something we want in our database?

#otherwise, create the new movie in the database

try:

data = request.get\_json() #grab any json data from the body of the request made to this route

new\_movie = Movies(data)

db.session.add(new\_movie)

db.session.commit()

return jsonify({'Created New Movie': new\_movie.to\_dict()})

except:

return jsonify({'Create Movie Rejected': 'Movie already exists or improper request.'})

#we cant test this create animals route because ['GET'] method doesnt exist for the url

#create animal only the ['POST']. to test we would have to write the request in python or use an API

#testing tool

C.) head over to a jupyter notebook to check the python way(reference the jupyter notebook you made)

D.) postman is an api testing tool that you can use alternatively to test your api post.head to the website account you created.

Tip:add some status codes by these routes. Reference the vs code document api routes

-create new data(create a new movie that’s not on the list)

the init for this is going to be different than the other table for the user because we are going to let some users create new data in our database and create a new animal(the user will be sending the 'POST' request in Json formatt to our system which translates into a python dictionary). Therefore, our init method for creating a new animal has to accept a single dictionary.

-delete data

#route for deleting a movie

@api.route('/movie/remove/<string:id>', methods=['DELETE'])

def removeMovie(id):

#if the movie is present in the database say so, and remove

movie = Movies.query.get(id)

if not movie: #if no movie with that id is in the database

#tell the user remove failed

return jsonify({'Remove failed':'No movie of that ID exists in the database'}), 404

db.session.delete(movie)

db.session.commit()

return jsonify({'Removed movie': movie.to\_dict()}), 200

-a dynamic route for getting a subset of data (ex. Film with a particular rating)

Tokenization

Head over to the api folder. Create new folder called services.py for the api tokenization

from app.models import User

from functools import wraps

#general structure of a custom decorator:

#outer function - name of the custom decorator

#wraps

#inner function

#stuff to run before decorated fucntion runs

#retuns decorated function

#returns inner function

def token\_required(api\_route):

@wraps(api\_route)

def decorator\_function(\*args, \*\*kwargs):

#the code here will run before the decorated route runs

#get the api token form the request headers

token = request.headers.get('x-access-token')

#if there is no token, stop the request from going through and send them a forbidden response

if not token:

return jsonify({'Access denied': 'No API token provided-please register an account and request an API token.'}), 401

#otherwise a token was provided but might not be valid.

if User.query.filter\_by(api\_token=token).first():

return jsonify({'Access denied': 'Invalid API token-please register an account and request an API token.'}), 403

return api\_route(\*args, \*\*kwargs)

return decorator\_function

1. Apply this token required decorator to our functions and test it

Over to routes.py

Type: #general structure of a custom decorator:

#outer function - name of the custom decorator

#wraps

#inner function

#stuff to run before decorated fucntion runs

#retuns decorated function

#returns inner function

def token\_required(api\_route):

@wraps(api\_route)

def decorator\_function(\*args, \*\*kwargs):

#the code here will run before the decorated route runs

#get the api token form the request headers

token = request.headers.get('x-access-token')

#if there is no token, stop the request from going through and send them a forbidden response

if not token:

return jsonify({'Access denied': 'No API token provided-please register an account and request an API token.'}), 401

#otherwise a token was provided but might not be valid.

if User.query.filter\_by(api\_token=token).first():

return jsonify({'Access denied': 'Invalid API token-please register an account and request an API token.'}), 403

return api\_route(\*args, \*\*kwargs)

return decorator\_function Make it so that users can have access tokens

@token\_required()

In the routes folder

from .services import token\_required