USER SETUP/ SET UP STEPS	Notes	completion
Setup Steps		
1. Download and extract the zip file from the starter code repo, open in VS Code.	flask tutorial 1	done
2. Create your own Github repo with a Python gitignore and your own README, then push the code up.	git hub repo has been created and all the files moved.	done
3. Create a MySQL Database with an appropriate name for the project.	flask tutorial 1	done
4. Change the username, password and db_name in the .env file to your MySQL username/password and the database you just created.	done: note, make sure that password is typed in without the <>, this held up last time.	done
NOTE: Some special characters in the password (especially @ and /) must be url escaped. For a @, use %40. For a /, use %2F	none	done
5. Create your virtual environment with terminal commands:	flask tutorial 1	done
pipenv install	flask tutorial 1	done
pipenv shell	flask tutorial 1	done
6. Once the venv is created, you can start the app at any time with flask run		
7. Create your database model(s) in app.py with the required properties, then run:		
flask db init (Creates tables)	make sure to do these before you get too far out; struggled through this a little, but got through it	done
flask db migrate -m "Init" (Creates migration)		done
flask db upgrade (Runs migration)		done
8. Create Marshmallow Schema in app.py		Done
Continue on to create Resource classes and Routes, making sure to test each endpoint in Postma		

TASKS/ USER STORIES	Notes	completion
(5 points): As a developer, I want to make good, consistent commits.	**created the init. Will commit after each story.**	**in progress**
(5 points): As a developer, I want to create an ERD for the Song Model, showing proper datatypes.	**done. Uploaded into repo.** **Update 04/26: edited ERD to reflect likes - Integer**	**done**

(2.5 points): As a developer, I want to create a Song model with the following properties. Property names must be in snake_case and match the following exactly! id - Integer title - String artist - String album - String release_date - Date genre - String	This has been completed based on the requirements. Note for Date the ANSI yyyy-mm-dd is used. To add in likes added in likes = db.Column(db.Integer, default = 0)	done
(5 points): As a developer, I want to create a SongSchema with a create_song() method so I can validate Posted songs.	@post_load def create_song(self, data, **kwargs): return Song(**data)	Done
(2.5 points): As a developer, I want my API to serve content on the following URLs paths: Paths must match these exactly! 'api/songs' 'api/songs/<int:pk>'	api.add_resource(SongResource, '/api/songs/<int:song_id>') api.add_resource(SongListResource, '/api/songs')	Done
(15 points): As a developer, I want to build a REST web API in Flask, so that I can make HTTP requests interact with the data set.	the rest web API in flask update 04/26: had some tables to clean after testing, and edited tables and cleaned them up using POSTMAN	
(5 points): As a developer, I want to create a GET endpoint that responds with a 200 success status code and all of the songs within the Song table.	get ALL was created in postman and executed. Status code in response area in Postman update 04/26: used get all to review current list. Changes to be made id 1: change genre to rock, id 2 change genre to pop, delete song 3, add back in the patsy cline song, confirm likes function works. no change to this...	done
(5 points): As a developer, I want to create a GET by id endpoint that does the following: Accepts a value from the request's URL (The id of the song to retrieve). Returns a 200 status code. Responds with the song in the database that has the id that was sent through the URL.	successfully executed the get by ID command: in postman entered in http://127.0.0.1:5000/api/songs/3 the 3 was id 3 and returned all song information in the response window. It was Patsy Cline, tennessee waltz Update 04/26: used get by id commands to grab song info prior to changing. no change made to this	done

(5 points): As a developer, I want to create a POST endpoint that does the following:
Accepts a body object from the request in the form of a Song model.
Validates the body object.
Adds the new song to the database.
Returns a 201 status code.
Responds with the newly created song object.
	entered the following into the body, raw, Json:
	{
	"artist": "simon_and_garfunkel",
	"release_date": "1970-01-26",
	"album": "bridge_over_troubled_water",
	"title": "the_only_living_boy_in_new_york",
	"genre": "pop"
	}
	
	201 status with CREATED shows in Postman, validated in mysql
	
	Update 04/26: when adding in the likes column needed the post to default likes to 0
		done

(5 points): As a developer, I want to create a PUT endpoint that does the following: Accepts a value from the request's URL (The id of the song to be updated). Accepts a body object from the request in the form of a Song model. Finds the song in the Song table and updates that song with any properties that were sent in the request's body. Returns a 200 status code. Responds with the newly updated song object.	in Postman, entered in PUT with http://127.0.0.1:5000/api/songs/1 NOTE: the change is dependent on having a field and the int in the http: after different rounds of testing, discovered the hard way and I overwrote the previous id 3 which was Patsy Cline's Tennesee Waltz. it took on hook with one of the updates i was making to the genre, copied and pasted all current information then put 3 in the http and it overwrote it completely. with the change to the genre and left previous still in id 1 update 04/26: for design purposes decided to return error for any puts on the likes column. so no updating any likes unless through mysql	done
(5 points): As a developer, I want to create a DELETE endpoint that does the following: Accepts a value from the request's URL. Deletes the object from the database. Returns a 204 (NO CONTENT) status code.	In postman entered in http://127.0.0.1:5000/api/songs/6;hit send. Message as defined from Flask provided no content and a 204 Status code. Validated in MySql update 04/26: validated still working	done
(5 points): As a developer, I want to use Postman to make a POST, PUT, DELETE, and both GET requests (get by id and get all) request to my REST web API, save it to a collection, and then export it as a JSON file from Postman. Include this file in your repo!	exported the collection as a JSON file and sent it to repository by doing a right click on the music library api … and exported into the repo, it is visible in VS Code. Udpate: 04/26: used POSTMAN to clean up the table and to do testing. Postman is functional	done

Bonus Stories **(5 points): As a developer, I want to add the ability to "like" a song through the web API and have the number of likes saved in the database with the song.**	**update 04/26: decided to go with the patch button after reviewing the definition of patch and seemed cleaner then creating two types of put commands if possible. In the development, decided to make it so that like was not editable through puts but could be increased through PATCH like by song which was not able to work if column initially starts off as null. needed to add default value to the db.**	**done**

CHECKLIST	**Notes**	**completion**

END RESULT	**Notes**	**completion**
The result of your Products API backend will be the execution of requests made in Postman. You must test your Products API by executing each request you create in Postman.	Postman is fully operational	done