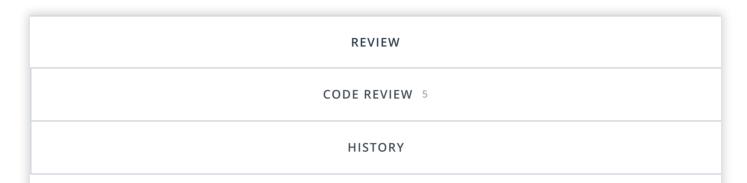


< Return to Classroom

Coffee Shop Full Stack



Meets Specifications

Congratulations, your project is successfully meeting all the requirements :-) Besides the specs, I want to highlight the following aspects of your project:

- Kudos for using abort() to properly raise HTTP errors
- I liked how you implemented guards within the PATCH and DELETE endpoints to guard against empty drinks manipulation, establishing a fail-fast strategy.

If you need further technical support focused on your project & lessons, feel free to open a new question on the Knowledge Hub.

Keep up the good work and good luck with the next project!

Flask server setup

All project code has been included in a single zip file.

The virtual env directory, pycache, and other local files are included in .gitignore .

Local files and virtual environment are included in .gitignore file.

TIPS & REFERENCES

Take a look at this StackOverflow's thread explaining in-depth the details and advantages of .gitgnore.

The code adheres to the PEP 8 style guide and follows common best practices, including:

- Variable and function names are clear.
- Endpoints are logically named.
- Code is commented appropriately.
- The README file includes detailed instructions for scripts to install any project dependencies, and to run the development server.
- Secrets are stored as environment variables.

Congratulations, your code is compliant with the best practices for python!

Clean Code:

Variable names, functions, and endpoints are named in a consistent and meaningful manner.



The README file is available and contains the required information about how to run your Flask backend.

Secrets:

Secrets are properly stored as environment variables.

TIPS & REFERENCES

• Run the pycodestyle command tool with the commands below:

```
pip install pycodestyle
pycodestyle [script_name].py
```

• Use autopep to automatically fix warnings.

All **@TODO** flags in the ./backend/src/api.py file have been completed.

The endpoints follow flask design principles, including <code>@app.route</code> decorators and request types.

The routes perform CRUD methods on the SQLite database using the simplified interface provided.

Best efforts should be made to catch common errors with @app.errorhandler decorated functions.

The following endpoints are implemented:

- GET /drinks
- GET /drinks-detail
- POST /drinks
- PATCH /drinks/<id>
- DELETE /drinks/<id>

It required endpoints marked with @TODO are implemented in the api.py file. If required endpoints marked with @TODO are implemented in the api.py file. If required HTTP handlers are implemented. AuthError handler he required AuthError handler is implemented. The backend can be run with the flask run and responds to all required REST requests. The backend can be run with no issues and all the exceptions are properly handled. IPS & REFERENCES You can start your Flask application with all the environment variables in a single line command: export FLASK_APP=api.py && export FLASK_ENV=development && python3 app.py PS: If you are using a Windows machine, just replace the export with set . Pomain Pame The Auth0 Domain Name The Auth0 Client ID It was provided a custom Auth0 Domain The JWT algorithm is properly parsing the tokens The API audience and configuration are functional	Well done! The flask	design principles were followed using the decorators and interfaces as required.
If required endpoints marked with @TODO are implemented in the api,py file. If HTTP Error handlers If required HTTP handlers are implemented. AuthError handler the required AuthError handler is implemented. AuthError handler the required AuthError handler is implemented. The backend can be run with the transplant of the backend can be run with no issues and all the exceptions are properly handled. IPS & REFERENCES You can start your Flask application with all the environment variables in a single line command: export FLASK_APP=ap1.py && export FLASK_ENV=development && python3 app.py PS: If you are using a Windows machine, just replace the export with set. PECURE A REST API for applications autho is set up and running at the time of submission. If required configuration settings are included in the auth.py file: The Autho Domain Name The Autho Client ID It was provided a custom Autho Domain The JWT algorithm is properly parsing the tokens The JWT algorithm is properly parsing the tokens The API audience and configuration are functional	Endpoints	
AuthError handler he required AuthError handler is implemented. AuthError handler he required AuthError handler is implemented. The backend can be run with flask run and responds to all required REST requests. The backend can be run with no issues and all the exceptions are properly handled. IPS & REFERENCES You can start your Flask application with all the environment variables in a single line command: export FLASK_APP=ap1.py && export FLASK_ENV=development && python3 app.py PS: If you are using a Windows machine, just replace the export with set. PECURE A REST API for applications AuthO is set up and running at the time of submission. AuthO is set up and running at the time of submission. The AuthO Domain Name The AuthO Client ID AuthO is up and running, good job! It was provided a custom AuthO Domain The JWT algorithm is properly parsing the tokens The API audience and configuration are functional		nts marked with @TODO are implemented in the api.py file.
AuthError handler he required AuthError handler is implemented. AuthError handler he required AuthError handler is implemented. The backend can be run with flask run and responds to all required REST requests. The backend can be run with no issues and all the exceptions are properly handled. IPS & REFERENCES You can start your Flask application with all the environment variables in a single line command: export FLASK_APP=ap1.py && export FLASK_ENV=development && python3 app.py PS: If you are using a Windows machine, just replace the export with set. PECURE A REST API for applications AuthO is set up and running at the time of submission. AuthO is set up and running at the time of submission. The AuthO Domain Name The AuthO Client ID AuthO is up and running, good job! It was provided a custom AuthO Domain The JWT algorithm is properly parsing the tokens The API audience and configuration are functional	LITTO Face a la con-	dlana
AuthError handler he required AuthError handler is implemented. he backend can be run with flask run and responds to all required REST requests. he backend can be run with no issues and all the exceptions are properly handled. IPS & REFERENCES • You can start your Flask application with all the environment variables in a single line command:		
the required AuthError handler is implemented. The backend can be run with flask run and responds to all required REST requests. The backend can be run with no issues and all the exceptions are properly handled. The backend can be run with no issues and all the exceptions are properly handled. The backend can be run with no issues and all the exceptions are properly handled. The backend can be run with no issues and all the exceptions are properly handled. The secure Flask application with all the environment variables in a single line command: Export Flask APP=ap1, py && export Flask ENV=development && python3 app, py PS: If you are using a Windows machine, just replace the export with set Becure a REST API for applications The Auth0 is set up and running at the time of submission. The Auth0 is set up and running at the time of submission. The Auth0 is up and running, good job! The JWT algorithm is properly parsing the tokens The API audience and configuration are functional	arrequired fiftr in	andiers are implemented.
he backend can be run with no issues and all the exceptions are properly handled. IPS & REFERENCES • You can start your Flask application with all the environment variables in a single line command: export FLASK_APP=ap1.py && export FLASK_ENV=development && python3 app.py PS: If you are using a Windows machine, just replace the export with set. Pecure a REST API for applications auth0 is set up and running at the time of submission. Ill required configuration settings are included in the auth, py file: • The Auth0 • Domain Name • The Auth0 Client ID auth0 is up and running, good job! It was provided a custom Auth0 Domain The JWT algorithm is properly parsing the tokens The API audience and configuration are functional		
PS & REFERENCES • You can start your Flask application with all the environment variables in a single line command: export FLASK_APP=api.py && export FLASK_ENV=development && python3 app.py PS: If you are using a Windows machine, just replace the export with set. Pecure a REST API for applications auth0 is set up and running at the time of submission. All required configuration settings are included in the auth.py file: • The Auth0 • Domain Name • The Auth0 Client ID auth0 is up and running, good job! It was provided a custom Auth0 Domain The JWT algorithm is properly parsing the tokens The API audience and configuration are functional	he backend can b	e run with flask run and responds to all required REST requests.
You can start your Flask application with all the environment variables in a single line command: export FLASK_APP=api.py && export FLASK_ENV=development && python3 app.py PS: If you are using a Windows machine, just replace the export with set. Pecure a REST API for applications auth0 is set up and running at the time of submission. Ill required configuration settings are included in the auth.py file: The Auth0 Domain Name The Auth0 Client ID It was provided a custom Auth0 Domain The JWT algorithm is properly parsing the tokens The API audience and configuration are functional	he backend can be	run with no issues and all the exceptions are properly handled.
export FLASK_APP=api.py && export FLASK_ENV=development && python3 app.py PS: If you are using a Windows machine, just replace the export with set. ecure a REST API for applications auth0 is set up and running at the time of submission. All required configuration settings are included in the auth.py file: The Auth0 Domain Name The Auth0 Client ID It was provided a custom Auth0 Domain The JWT algorithm is properly parsing the tokens The API audience and configuration are functional	TPS & REFERENCES	
export FLASK_APP=api.py && export FLASK_ENV=development && python3 app.py PS: If you are using a Windows machine, just replace the export with set. ecure a REST API for applications auth0 is set up and running at the time of submission. All required configuration settings are included in the auth.py file: The Auth0 Domain Name The Auth0 Client ID It was provided a custom Auth0 Domain The JWT algorithm is properly parsing the tokens The API audience and configuration are functional	• You can star	t your Flask application with all the environment variables in a single line command:
ecure a REST API for applications auth0 is set up and running at the time of submission. all required configuration settings are included in the auth.py file: The Auth0 Domain Name The Auth0 Client ID auth0 is up and running, good job!		· · · · · · · · · · · · · · · · · · ·
auth0 is set up and running at the time of submission. Ill required configuration settings are included in the auth.py file: The Auth0 Domain Name The Auth0 Client ID It was provided a custom Auth0 Domain The JWT algorithm is properly parsing the tokens The API audience and configuration are functional	PS: If you are	e using a Windows machine, just replace the export with set .
It was provided a custom Auth0 Domain The JWT algorithm is properly parsing the tokens The API audience and configuration are functional	Auth0 is set up and All required configu • The Auth0 • Domain Nar • The Auth0 C	I running at the time of submission. Uration settings are included in the auth.py file: me lient ID
The JWT algorithm is properly parsing the tokens The API audience and configuration are functional	Auth0 is up and run	ning, good job!
The API audience and configuration are functional	🗸 lt was provided	a custom Auth0 Domain
	The JWT algorith	nm is properly parsing the tokens
	The API audiend	ce and configuration are functional
custom @requires_auth decorator is completed in ./backend/src/auth/auth.py	0. 20042 11 2	

The @requres_auth decorator should:

- Get the Authorization header from the request.
- Decode and verify the JWT using the Auth0 secret.
- Take an argument to describe the action (i.e., <code>@require auth('create:drink')</code>).
- Raise an error if:
 - The token is expired.
 - The claims are invalid.
 - The token is invalid.
 - The JWT doesn't contain the proper action (i.e. create: drink).

The code is neat and the common JWT errors were caught!

- The application can fetch the token from the HTTP request header.
- Can check the signature using dynamic values like Auth0 secret and audience.
- Raise errors if any of the common scenario validation fails.
- The design pattern of decorators was well applied to be used within routes.

TIPS & REFERENCES

 You can learn better about Flask Decorators here and implement custom decorators for other purposes.

Roles and permission tables are configured in Autho. The JWT includes the RBAC permission claims.

Barista access is limited:

- · can get drinks
- can get drink-details

Manager access is limited

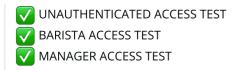
- · can get drinks
- · can get drink details
- · can post drinks
- can patch drinks
- can delete drinks

The provided postman collection passes all tests when configured with valid JWT tokens.

You must export the postman collection to

./starter_code/backend/udacity-fsnd-udaspicelatte.postman_collection.json with your JWTs configured by right-clicking the collection folder for barista and manager, navigating to the authorization tab, and including the JWT in the token field.

All the roles and permissions were properly implemented, congratulations! The postman collections are working fine with the provided tokens and all tests passed:



TIPS & REFERENCES

• The POSTman suit is a fast-growing tool with several capabilities, and they are always releasing new cool stuff. I advise you to visit the POSTman learning center to raise your awareness of what is possible with this fantastic tooling set.

Front end

The frontend has been configured with Auth0 variables and backend configuration.

The ./frontend/src/environment/environment.ts file has been modified to include the student's variables.

All required variables are available in the front-end environment; therefore, the application build and run as expected!

TIPS & REFERENCES

• You can read this article to learn how helpful is this configuration file when you are using multiple environments like development, testing, pre-production, production, demo, etc.

The frontend can be run locally with no errors with ionic serve and displays the expected results.

The ionic frontend is running properly with the expected results on UI!



>

CODE REVIEW COMMENTS

RETURN TO PATH

Rate this review

START