UDACITY

Logout

PROJECT SPECIFICATION

Coffee Shop Full Stack

Flask server setup

CRITERIA	MEETS SPECIFICATIONS
The complete project has been submitted as a zip and demonstrates the ability to share code on git.	All project code has been included in a single zip file. The virtual env directory, pycache, and other local files are included in `.gitignore`.
The project demonstrates coding best practices.	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.
The project demonstrates an understanding of restful APIs.	All `@TODO` flags in the *./backend/src/api.py* file have been completed. The endpoints follow flask design principles, including `@app.route` 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>`
The project demonstrates the ability to build a functional backend.	The backend can be run with `flask run` and responds to all required REST requests.

Secure a REST API for applications

CRITERIA	MEETS SPECIFICATIONS
The project demonstrates an understanding of third-party	Auth0 is set up and running at the time of submission.

authentication systems.	All required configuration settings are included in the `auth.py` file: • The Auth0 • Domain Name • The Auth0 Client ID
The project demonstrates an understanding of JWTs and Role Based Authentication.	A custom `@requires_auth` decorator is completed in `./backend/src/auth/auth.py` 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., `@require_auth('create:drink'`). • 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 project demonstrates the ability to secure a system through an understanding of roles-based access control (RBAC) .	Roles and permission tables are configured in Auth0. 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.

Front end

CRITERIA	MEETS SPECIFICATIONS
The project demonstrates an understanding of how to loosely uncouple authentication and REST services.	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.
The project demonstrates the	The frontend can be run locally with no errors with `ionic serve` and displays the expected results

The project demonstrates the ability to work across the stack.	The frontend can be run locally with no errors with `ionic serve` and displays the expected results.

Suggestions to Make Your Project Stand Out!

1. Create endpoints to manage users using the Auth0 API
2. Barista access is limited (can do nothing)
3. Manager access is limited (can manage baristas)
4. Administrator access is limited (can manage baristas, managers)
5. Deploy the service to a cloud provider such as elastic beanstalk or Heroku
6. Configure Auth0 with multi-factor authentication or other social OpenIDs
7. Modify the front end with some unique styles or functionality