Coffee API

Release 1.0

Scott R Smith

Dec 10, 2019

CONTENTS

CHAPTER

ONE

INSTALATION AND OVERVIEW

1.1 Intoduction

The coffee shop app is a new digitally enabled cafe for udacity students to order drinks, socialize, and study hard. The full stack drink menu application does the following:

1) Display graphics representing the ratios of ingredients in each drink.
2) Allow public users to view drink names and graphics.
3) Allow the shop baristas to see the recipe information.
4) Allow the shop managers to create new drinks and edit existing drinks.

1.2 Getting Started - Backend

1.2.1 Installing Dependencies

Python 3.7

This project uses python 3.7.

To Install: Python

1.2.1.1 PIP Dependencies

Once you have your virtual environment setup and running, install dependencies by navigating to the root directory and running:

```
pip install -r requirements.txt
```

This will install all of the required packages we selected within the `requirements.txt` file.

Key Dependencies

- Flask is a lightweight backend microservices framework.
- SQLAlchemy is the Python SQL toolkit and ORM.
- Flask-CORS is the extension used to handle cross-origin requests from the frontend server.
- Auth0 Provides authentication and authorization as a service

1.3 Database Setup

The app is running with SQLite. No setup needs to be performed.

1.4 Running the server

From within the `backend/src` directory to run the server, execute:

```
export FLASK_APP=api.py
export FLASK_ENV=development
flask run
```

1.5 Documentation

1.5.1 Opening the API Documentation

Documentation is generated with Sphinx.

1.5.2 HTML Documentation

From the root folder, open the index file in a browser

```
./docs/build/html/index.html
```

1.5.2.1 PDF Documentation

The PDF version of the documentation is located in the root project directory. Named coffeeapi.pdf

1.5.3 Generating documentation

Documentation is generated with Sphinx.

1.5.3.1 Installing Sphinx and support tools

To install Sphinx, reference the documents at https://www.sphinx-doc.org/en/master/usage/installation.html

For example:

```
pip install -U sphinx
```

Install dependencies by navigating to the `root` project directory and running:

```
cd docs
pip install m2r
pip install recommonmark
pip install rinohtype
pip install -r requirements.txt
```

1.5.3.2 Generating the documentation

Generate the documentation with the following commands:

```
# From the root project directory
# Convert readme to rst to be included in generated docs
m2r README.md README.rst --overwrite
cp -R README.rst ./docs/source
cd ./docs
make html
# Make pdf
make latexpdf
cd ..
cp -R ./docs/build/latex/coffeeaapi.pdf .
```

1.6 API End Points

The following APIs are available. Detailed html documentation can be found in the 'docs' folder.

- GET /drinks
- GET /drinks-detail
- POST /drinks
- PATCH /drinks/
- DELETE /drinks/

1.7 Error Handling

Errors are returned as JSON objects in the following format:

```
{
    "success": False,
    "error": 400,
    "message": "Bad Request"
}
```

The API will return three error types when requests fail:

- 400: Bad Request
- 404: Resource Not Found
- 405: Method Not Allowed
- 422: Not Processable
- 500: Internal Server Error

1.8 Testing

Testing is done with Postman. Load and run the test collection: .backend/udacity-fsnd-udaspicelatte.postman_collection.json

1.9 Full Stack coffee API Frontend

1.9.1 Installing Dependencies

1.9.1.1 Installing Node and NPM

This app depends on Nodejs and Node Package Manager (NPM). Before continuing, you must download and install Node (the download includes NPM) from https://nodejs.com/en/download.

1.9.1.2 Installing Ionic Client

The Ionic Command Line Interface is required to serve and build the frontend. Instructions for installing the CLI is in the Ionic Framework Docs.

1.9.1.3 Installing project dependencies

This project uses NPM to manage software dependencies. NPM Relies on the package.json file located in the `frontend` directory of this repository. After cloning, open your terminal and run:

```
npm install
```

1.9.1.4 Running the Frontend

To run Ionic from the `frontend` directory run:

```
ionic serve
```

CHAPTER

TWO

COFFEE API CONTROLLERS

2.1 Introduction

The coffee shop app is a new digitally enabled cafe for udacity students to order drinks, socialize, and study hard. The full stack drink menu application with the following endpoints implemented:

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

With the following API permissions: - *get:drinks-detail* (Barista and Manager) - *post:drinks* (Manager) - *patch:drinks* (Manager) - *delete:drinks* (Manager)

src.api.**list_of_drinks_short_form**()

Get Drinks

Get a list of the drinks in short form

Get Drinks

- Sample Call create question:

```
curl -X POST http://localhost:5000/drinks
     -H 'content-type: application/json'
```

- Expected Success Response:

```
HTTP Status Code: 200

{
 "success": true,
 "drinks": [ list of drink.short() ]
}
```

- Expected Fail Response:

```
HTTP Status Code: 401
{
    "description": "401: Authorization header is expected.",
    "error": 401,
    "message": "Unauthorized",
```

(continues on next page)

(continued from previous page)

```
    "success": false
}
```

src.api.**list_of_drinks_long_form**(*payload*)

Retrieve Drink Details in Long form

Get a list of the drinks in long form

Get Drinks

- Sample Call create question:

```
curl -X POST http://localhost:5000/drinks
     -H 'content-type: application/json'
```

- Expected Success Response:

```
HTTP Status Code: 200

{
"drinks": [
    {
    "id": 1,
    "recipe": [
        {
        "color": "white",
        "name": "Vodka",
        "parts": 1.5
        },
        {
        "color": "red",
        "name": "Cranberry Juice",
        "parts": 1
        }
    ],
    "title": "Cosmo"
    }
],
"success": true
}
```

- Expected Fail Response:

```
HTTP Status Code: 401
{
    "description": "401: Authorization header is expected.",
    "error": 401,
    "message": "Unauthorized",
    "success": false
}
```

src.api.**create_new_drink**(*payload*)

Create Drink

This API will create a new Drink.

Create a new Drink

- Sample Call create drink:

```
curl -X POST http://localhost:5000/drinks                    -H 'content-type:␣
↪application/json'                  -d '{"title": "Water3",                    ␣
↪     "recipe": [{"name": "Water",                                  "color":
↪"blue",                                  "parts": 1                          ␣
↪        }]                    }'
```

- Expected Success Response:

```
HTTP Status Code: 200

{
"drinks": {
    "id": 2,
    "recipe": [
    {
        "color": "blue",
        "name": "water",
        "parts": 1
    }
    ],
    "title": "Water"
},
"success": true
}
```

- Expected Fail Response:

```
HTTP Status Code: 401
{
    "description": "401: Authorization header is expected.",
    "error": 401,
    "message": "Unauthorized",
    "success": false
}
```

src.api.**update_drink** (*payload*, *drink_id*)

Updates a drink

This API will update a drink by drink Id.

- Sample Call:

```
curl -X PATCH http://localhost:5000/drink/1                  -H 'content-
↪type: application/json'                  -d '{"title": "water"}'
```

- Expected Success Response:

```
HTTP Status Code: 200

{
"drinks": [
    {
    "id": 1,
    "recipe": [
        {
        "color": "blue",
        "name": "water",
        "parts": 1
```

(continues on next page)

(continued from previous page)

```
        }
    ],
    "title": "water"
    }
],
"success": true
}
```

- Expected Fail Response:

```
HTTP Status Code: 401
 {
    "description": "401: Authorization header is expected.",
    "error": 401,
    "message": "Unauthorized",
    "success": false
}
```

src.api.**delete_drink**(*payload*, *drink_id*)

Delete a drink from the database

This API will delete a drink from the database.

- Sample Call:

```
curl -X DELETE http://localhost:5000/drink/2
     -H 'content-type: application/json'
```

- Expected Success Response:

```
HTTP Status Code: 200
{
    "deleted": 2,
    "success": true
}
```

- Expected Fail Response:

```
HTTP Status Code: 401
{
    "description": "401: Authorization header is expected.",
    "error": 401,
    "message": "Unauthorized",
    "success": false
}
```

CHAPTER

THREE

COFFEE API MODEL

Introduction

The Coffee shop app includes uses a single Alchemy classes to manage Drinks.

- Drink Class : a persistent drink entity, extends the base SQLAlchemy Model.

The drink class has the following attributes:

- id: The auto-generated record ID
- title: The name of the drink, unique
- List of ingredients, as an json object like [{'color': string, 'name':string, 'parts':number}]

class src.database.models.**Drink**(*title*, *recipe*)

A persistent drink entity, extends the base SQLAlchemy Model

id

id is the auto assigned primary key. Type: Integer, Primary key. Required.

title

title, String(80) The title (name) of the drink

recipe

recipe, string(160) The ingredients blob - this stores a lazy json blob The required datatype is [{'color': string, 'name':string, 'parts':number}]

short()

short form representation of the Drink model

Returns:

```
{ 'id': self.id,
  'title': self.title,
  'recipe': [{'color': string,
              'name':string,
              'parts':number
             }]
}
```

long()

long form representation of the Drink model.

Returns:

```
{ 'id': self.id,
  'title': self.title,
```

(continues on next page)

(continued from previous page)

```
  'recipe': self.recipe
}
```

insert()
Inserts a new model into a database. The model must have a unique id and title.

EXAMPLE:

```
drink = Drink(title=req_title, recipe=req_recipe)
drink.insert()
```

delete()
deletes a new model into a database the model must exist in the database

EXAMPLE:

```
drink = Drink(title=req_title, recipe=req_recipe)
drink.delete()
```

update()
updates a new model into a database the model must exist in the database

EXAMPLE:

```
drink = Drink.query.filter(Drink.id == id).one_or_none()
drink.title = 'Black Coffee'
drink.update()
```

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

INDEX