Coffee Shop Full Stack

REVIEW | CODE REVIEW 8 | HISTORY

backend/src/api.py 5

```python
import json
import sys

from flask import Flask, request, jsonify, abort
from flask_cors import CORS

from .auth.auth import AuthError, RESPONSE_CODE, requires_auth, AUTH_HEADER_ERR
from .database.models import db_drop_and_create_all, setup_db, Drink

app = Flask(__name__)
setup_db(app)
CORS(app)

# db_drop_and_create_all()


# Get all drinks
@app.route("/", methods=["GET"])
def index():
    return "Open for business!"


# Get basic drink details
@app.route("/drinks", methods=["GET"])
def get_drinks():
    try:
        drinks = Drink.query.all()
        short_drinks = [drink.short() for drink in drinks]

        response = {
            "Success": True,
            "drinks": short_drinks
        }

        return jsonify(response), RESPONSE_CODE["200_OK"]

    except:
        print(sys.exc_info())
        abort(RESPONSE_CODE["500_INTERNAL_SERVER_ERROR"])


# Get full drink details
@app.route("/drinks-detail", methods=["GET"])
@requires_auth("get:drinks-detail")
```

AWESOME

Great job adding the `@requires_auth` decorator as the auth middleware on your routes. 🔒

```python
def get_drinks_detail(jwt):
    try:
        drinks = Drink.query.all()
        long_drinks = [drink.long() for drink in drinks]

        response = {
            "Success": True,
            "drinks": long_drinks
        }

        return jsonify(response), RESPONSE_CODE["200_OK"]

    except:
        print(sys.exc_info())
        abort(RESPONSE_CODE["500_INTERNAL_SERVER_ERROR"])


# POST a new drink
@app.route("/drinks", methods=["POST"])
@requires_auth("post:drinks")
def post_drink(jwt):
    body = request.get_json()

    if "title" not in body or "recipe" not in body:
```

```
    if "title" not in body or "recipe" not in body:
        abort(RESPONSE_CODE["422_UNPROCESSABLE_ENTITY"])

    drink_title = body.get("title", None)
    drink_recipe = body.get("recipe", None)

    # Instantiate new drink
    new_drink = Drink(
        title=drink_title,
        recipe=json.dumps(drink_recipe)
    )

    # Add new drink to db
    new_drink.insert()

    response = {
        "success": True,
        "drinks": [new_drink.long()],
    }

    return jsonify(response), RESPONSE_CODE["200_OK"]


# Update a drink
@app.route("/drinks/<int:drink_id>", methods=["PATCH"])
@requires_auth("patch:drinks")
def patch_drink(jwt, drink_id):
    try:
        # Error handling for id
        drink_count = len(Drink.query.all())
        if drink_id is None or drink_id <= 0 or drink_id > drink_count:
            abort(RESPONSE_CODE["404_RESOURCE_NOT_FOUND"])

        # Error handling for existing drink
        existing_drink = Drink.query.filter(Drink.id == drink_id).one_or_none()
        if existing_drink is None:
```

AWESOME

Great job validating that the drink does exist before editing it.

```
            abort(RESPONSE_CODE["404_RESOURCE_NOT_FOUND"])

        # Retrieving values for updating
        body = request.get_json()

        # Check if body contains any data
        if body is not False:
            drink_new_title = ""
            drink_new_recipe = ""

            # Retrieve and update the existing record with new data if provided
            if "title" in body:
                drink_new_title = body.get("title")
                existing_drink.title = drink_new_title
            if "recipe" in body:
                drink_new_recipe = body.get("recipe")
                existing_drink.recipe = json.dumps(drink_new_recipe)

            # Update record only if it's data has changed
            if existing_drink.title == drink_new_title or \
                    existing_drink.recipe == json.dumps(drink_new_recipe):
                existing_drink.update()

        response = {
            "success": True,
            "drinks": [existing_drink.long()],
        }

        return jsonify(response), RESPONSE_CODE["200_OK"]

    except:
        print(sys.exc_info())
        raise AuthError({
            "code": AUTH_HEADER_ERR["INVALID_HEADER"],
            "description": "Unauthorised"
        }, RESPONSE_CODE["401_UNAUTHORIZED"])


# Delete a drink
@app.route("/drinks/<int:drink_id>", methods=["DELETE"])
```

SUGGESTION

You can explore a little bit further the code documentation opportunity here. Flask API's has great integrations with the Swagger tool, but you do not need to go fancy on this one, just documenting the entry parameters and expected return would be great enough.

Check out this thread, which contains a ton of great tips and examples on how to enable your documentation to be read by swagger, and by the humans too!

```
@requires_auth("delete:drinks")
def delete_drink(jwt, drink_id):
    try:
        drink_count = len(Drink.query.all())

        if drink_id is None or drink_id <= 0 or drink_id > drink_count:
            abort(RESPONSE_CODE["404_RESOURCE_NOT_FOUND"])

        target_drink = Drink.query.filter(Drink.id == drink_id).one_or_none()
```

```
153
154         if target_drink is None:
155             abort(RESPONSE_CODE["404_RESOURCE_NOT_FOUND"])
156
157         target_drink.delete()
158
159         response = {
160             "success": True,
161             "delete": drink_id,
162         }
163
164         return jsonify(response), RESPONSE_CODE["200_OK"]
165     except:
166         print(sys.exc_info())
167         raise AuthError({
168             "code": AUTH_HEADER_ERR["INVALID_HEADER"],
169             "description": "Unauthorised"
170         }, RESPONSE_CODE["401_UNAUTHORIZED"])
171
172
173 # Error Handling
174 @app.errorhandler(RESPONSE_CODE["400_BAD_REQUEST"])
```

AWESOME

The required custom error handlers are in place, great job adding them!

```
175 def bad_request(error):
176     print(error)
177
178     return jsonify({
179         "success": False,
180         "error": RESPONSE_CODE["400_BAD_REQUEST"],
181         "message": "Bad request",
182     }), RESPONSE_CODE["400_BAD_REQUEST"]
183
184
185 @app.errorhandler(RESPONSE_CODE["404_RESOURCE_NOT_FOUND"])
```

AWESOME

I really liked the way you organized the flow of your app, it is very well organized and maintainable. Kudos! 👍🏽

```
186 def resource_not_found(error):
187     print(error)
188
189     return jsonify({
190         "success": False,
191         "error": RESPONSE_CODE["404_RESOURCE_NOT_FOUND"],
192         "message": "Resource not found",
193     }), RESPONSE_CODE["404_RESOURCE_NOT_FOUND"]
194
195
196 @app.errorhandler(RESPONSE_CODE["422_UNPROCESSABLE_ENTITY"])
197 def unprocessable_entity(error):
198     print(error)
199
200     return jsonify({
201         "success": False,
202         "error": RESPONSE_CODE["422_UNPROCESSABLE_ENTITY"],
203         "message": "Unprocessable entity",
204     }), RESPONSE_CODE["422_UNPROCESSABLE_ENTITY"]
205
206
207 @app.errorhandler(RESPONSE_CODE["500_INTERNAL_SERVER_ERROR"])
208 def internal_server_error(error):
209     print(error)
210
211     return jsonify({
212         "success": False,
213         "error": RESPONSE_CODE["500_INTERNAL_SERVER_ERROR"],
214         "message": "Internal server error",
215     }), RESPONSE_CODE["500_INTERNAL_SERVER_ERROR"]
216
217
218 @app.errorhandler(AuthError)
219 def auth_error(exception):
220     print(exception)
221
222     response = jsonify(exception.error)
223     response.status_code = exception.status_code
224     return response
225
```

▸ backend/src/auth/auth.py 3

▸ README.md

▸ backend/src/__init__.py

▸ backend/src/auth/__init__.py

▸ backend/src/database/__init__.py

- backend/src/database/models.py
- backend/README.md
- backend/requirements.txt
- frontend/src/karma.conf.js
- frontend/src/app/pages/drink-menu/drink-graphic/drink-graphic.component.html
- frontend/src/app/pages/drink-menu/drink-menu.page.html
- frontend/src/app/pages/drink-menu/drink-form/drink-form.component.html
- frontend/src/app/pages/user-page/user-page.page.html
- frontend/src/app/pages/tabs/tabs.page.html
- frontend/src/app/app.component.html
- frontend/src/index.html
- frontend/e2e/protractor.conf.js
- frontend/README.md

RETURN TO PATH