Coffee Shop Full Stack

REVIEW | CODE REVIEW 8 | HISTORY

▸ backend/src/api.py 5

▾ backend/src/auth/auth.py 3

```python
import json
from functools import wraps
from urllib.request import urlopen

from flask import request, abort
from jose import jwt

AUTH0_DOMAIN = "fsnd2020.auth0.com"
```

AWESOME

Auth0 information is in place, great job

```python
ALGORITHMS = ["RS256"]
API_AUDIENCE = "cafe"

RESPONSE_CODE = {
    # Success codes
    "200_OK": 200,

    # Client error codes
    "400_BAD_REQUEST": 400,
    "401_UNAUTHORIZED": 401,
    "403_FORBIDDEN": 403,
    "404_RESOURCE_NOT_FOUND": 404,
    "422_UNPROCESSABLE_ENTITY": 422,

    # Server error codes
    "500_INTERNAL_SERVER_ERROR": 500
}

AUTH_HEADER_ERR = {
    "INVALID_HEADER": "invalid_header",
    "INVALID_CLAIMS": "invalid_claims",
    "TOKEN_EXPIRED": "token_expired",
}


# AuthError Exception: A standardised way to communicate auth failure modes
class AuthError(Exception):
    def __init__(self, error, status_code):
        self.error = error
        self.status_code = status_code


# Retrieve token from auth header
def get_token_auth_header():
    if "Authorization" not in request.headers:
        raise AuthError({
            "code": AUTH_HEADER_ERR["INVALID_HEADER"],
            "description": "Authorization header is missing"
        }, RESPONSE_CODE["401_UNAUTHORIZED"])

    # Dissect header
    header_parts = request.headers["Authorization"].split()
    bearer_prefix = header_parts[0]
    auth_token = header_parts[1]

    # 401 err handling
    if not bearer_prefix:
        raise AuthError({
            "code": AUTH_HEADER_ERR["INVALID_HEADER"],
            "description": "Authorization header is missing bearer prefix"
        }, RESPONSE_CODE["401_UNAUTHORIZED"])

    elif bearer_prefix and len(header_parts) == 1:
        raise AuthError({
            "code": AUTH_HEADER_ERR["INVALID_HEADER"],
            "description": "Authorization header is missing auth token"
```

```
        }, RESPONSE_CODE["401_UNAUTHORIZED"])

    elif len(header_parts) > 2:
        raise AuthError({
            "code": AUTH_HEADER_ERR["INVALID_HEADER"],
            "description": "Authorization header is malformed"
        }, RESPONSE_CODE["401_UNAUTHORIZED"])

    return auth_token


def check_permissions(permission: str, payload):
    # Check if user has any permissions
    if "permissions" not in payload:
        abort(RESPONSE_CODE["400_BAD_REQUEST"])

    # Check if user has the requested permission
    elif permission not in payload["permissions"]:
        abort(RESPONSE_CODE["401_UNAUTHORIZED"])

    return True


def verify_decode_jwt(token: str):
    jsonurl = urlopen(f"https://{AUTH0_DOMAIN}/.well-known/jwks.json")
    jwks = json.loads(jsonurl.read())
    unverified_header = jwt.get_unverified_header(token)
    rsa_key = {}

    if "kid" not in unverified_header:
        raise AuthError({
            "code": AUTH_HEADER_ERR["INVALID_HEADER"],
            "description": "Authorization header is malformed"
        }, RESPONSE_CODE["401_UNAUTHORIZED"])

    for key in jwks["keys"]:
        if key["kid"] == unverified_header["kid"]:
            rsa_key = {
                "kty": key["kty"],
                "kid": key["kid"],
                "use": key["use"],
                "n": key["n"],
                "e": key["e"]
            }

    if rsa_key:
        try:
            payload = jwt.decode(
                token,
                rsa_key,
                algorithms=ALGORITHMS,
                audience=API_AUDIENCE,
                issuer=f"https://{AUTH0_DOMAIN}/"
            )

            return payload

        except jwt.ExpiredSignatureError:
            raise AuthError({
                "code": AUTH_HEADER_ERR["TOKEN_EXPIRED"],
                "description": "Token expired"
            }, RESPONSE_CODE["401_UNAUTHORIZED"])

        except jwt.JWTClaimsError:
            raise AuthError({
                "code": AUTH_HEADER_ERR["INVALID_CLAIMS"],
                "description": "Incorrect claims. Please, check the audience and issuer"
            }, RESPONSE_CODE["401_UNAUTHORIZED"])

        except Exception:
            raise AuthError({
                "code": AUTH_HEADER_ERR["INVALID_HEADER"],
                "description": "Unable to parse authentication token"
            }, RESPONSE_CODE["400_BAD_REQUEST"])

    raise AuthError({
        "code": AUTH_HEADER_ERR["INVALID_HEADER"],
        "description": "Unable to find the appropriate key"
    }, RESPONSE_CODE["400_BAD_REQUEST"])


def requires_auth(permission=""):
```

AWESOME

Decorator and wrapper also in place

```
    def requires_auth_decorator(f):
        @wraps(f)
        def wrapper(*args, **kwargs):
            # Retrieve jwt from auth header
```

AWESOME

Great job adding some comments which provides an entry-level of the application documentation; it is helpful to remember you in the future or even bring other developers to contribute to your source code.

```
            jwt = get_token_auth_header()

            # Grab the payload from decoded jwt
            try:
```

```
            try:
                payload = verify_decode_jwt(jwt)
            except:
                raise AuthError({
                    "code": AUTH_HEADER_ERR["INVALID_HEADER"],
                    "description": "Token unauthorised"
                }, RESPONSE_CODE["401_UNAUTHORIZED"])

            # Check if user has any / requested permission(s)
            check_permissions(permission, payload)

            return f(payload, *args, **kwargs)

        return wrapper

    return requires_auth_decorator

```

▸ 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