Team 7

SOFTWARE DESIGN DOCUMENT

GPYOU

Project Manager	Luv Shah
Design Manager	Kosta Kyriakoulis
Testing Manager	Dave Patel
Requirements Manager	David Kulis
Presentation Manager	Jayden Godbold

November 22, 2022

CONTENTS

INTRODUCTION

The intent of this document is to highlight noteworthy features of the GPYou system at play, by proving insight into how the low-level functions and design structures work and expose the way data percolates to and from each component of the system.

Some topics this document covers include the following:

- API Layers
- Backend to Frontend stack
- User interface design
- Test cases with expected results
- Processing Scenarios

With this documentation, the reader can understand the inner mechanisms in action for both administrative roles and ordinary users.

1.1 GOALS AND OBJECTIVES

The purpose of GPYou is to ultimately allow the user to collect a listing of Graphics Processing Units (GPU's) available through online commercial outlets such as e-commerce sites, online store fronts propriated by established retailers, and secondhand reseller sites for used products. The collection of listings generated by GPYou is made by taking user inputs to then feed into the back-end API.

With convenience in mind, the user should be able to use this product with little to no experience. The features available are presented in a format the intended user maybe familiar with. As such, this product is works with the user in delivering the necessary information quickly and without the need to filter through web-ads and superfluous promotions.

Other notable functions include:

- Administrator's Panel
- A user's homepage
- Results page for search results
- Favorites panel
- Search preferences menu
- Login and Registration page

1.2 CORE FEATURES

1. User Registration and Welcome
    - Allows user to sign up for becoming a user of the web app
    - Verifies password is 16-64 characters of alpha-numeric ASCII values
    - Verifies username is unique and available for the user to authenticate with
2. Login Page
    - Authenticated recurring users with their corresponding user accounts
    - Prevents unauthenticated users from accessing the functionalities of GPYou
3. Searching for a GPU
    - User can set a minimum and maximum price range for GPU's in USD
    - User can input a specific memory size the GPU must contain
    - User can input a desired manufacturer that fabricates a GPU model
    - User can manually type in a name for the desired GPU model
    - A search can be executed on one or more of the provided search terms
4. Results Page
    - A list of search results is displayed on a separate page rendered by the server
    - The listings are ordered by price in ascending order
    - The listings are presented as a grid, with each row containing an individual listing
    - Each row contains an unmarked check box toggle on or off as a Favorite
    - Links are cited for each row on the results table, redirecting to the source listing
5. Favorites Panel
    - The user can access their favorites list in a grid format like the results page
    - The favorites list contains links to the
    - The user can remove a listing on their favorites list
6. Administrators Panel
    - Only accessible by authenticating with admin credentials via Login page
    - Admin can designate any user with admin roles and privileges
    - Admin can delete a user from the GPYou registration
    - Admin can add a user to the GPYou registration
    - Admin can modify a user's username and password in the GPYou registration

TECHNICAL OVERVIEW

To ensure a robust and efficient experience for any user of this application, a set of test plans where designed around specific test cases the application should expect to incur.

A test suite which will run to make sure the intended functionality will perform as expected in a deterministic process. This will allow progress, implement new features, and help create a test-driven environment.

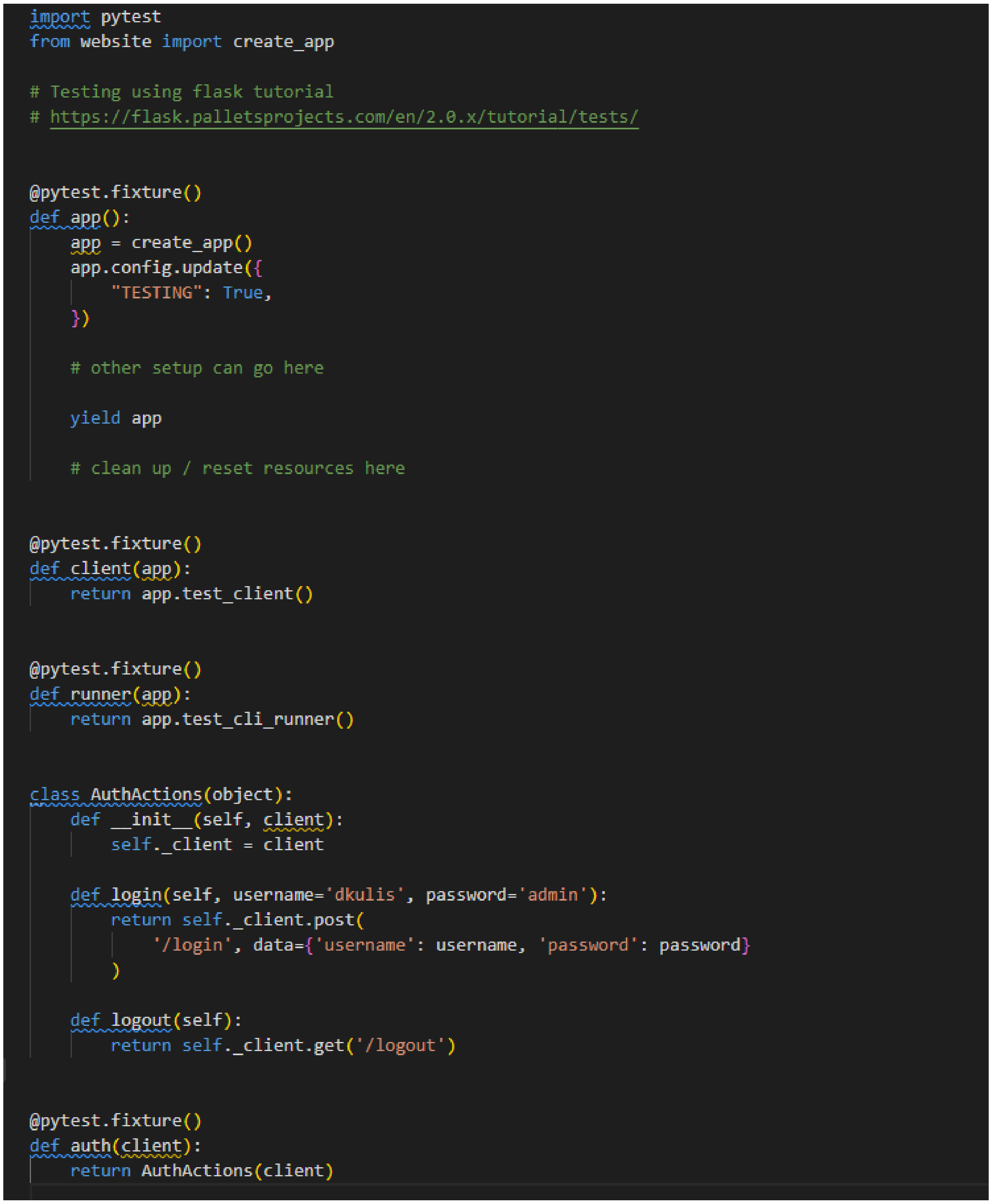
2.1 TEST PLAN

1. Blackbox testing Admin Panel
    - Correctly removing users from the database
    - Correctly adding users to database
    - Creating an admin user
    - Deleting an admin user
2. Database retrieval
    - Display parsed GPUs that match search term
    - Display correct favorites list for each user
3. Test Endpoints
    - Use sessions to check if currently passed in user is logged in
    - Test to see if each webpage is accessible
    - Show proper errors when logging in as a fake user
    - Show errors when registering as user that already exists

2.2 TEST CASES

The test suite mention before, is applied onto the GPYou’s Amazon Scrapper and parser which parsed the data from the scrapper, the GPYou’s backend database layer to see if data was properly inserted when registering a user, and ascertains routes such as: Login, register, and admin.

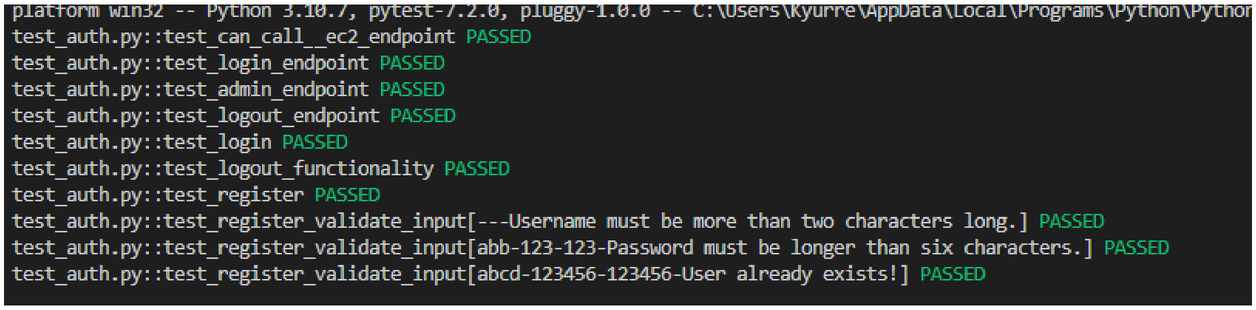
SETTING UP THE TEST SUITE



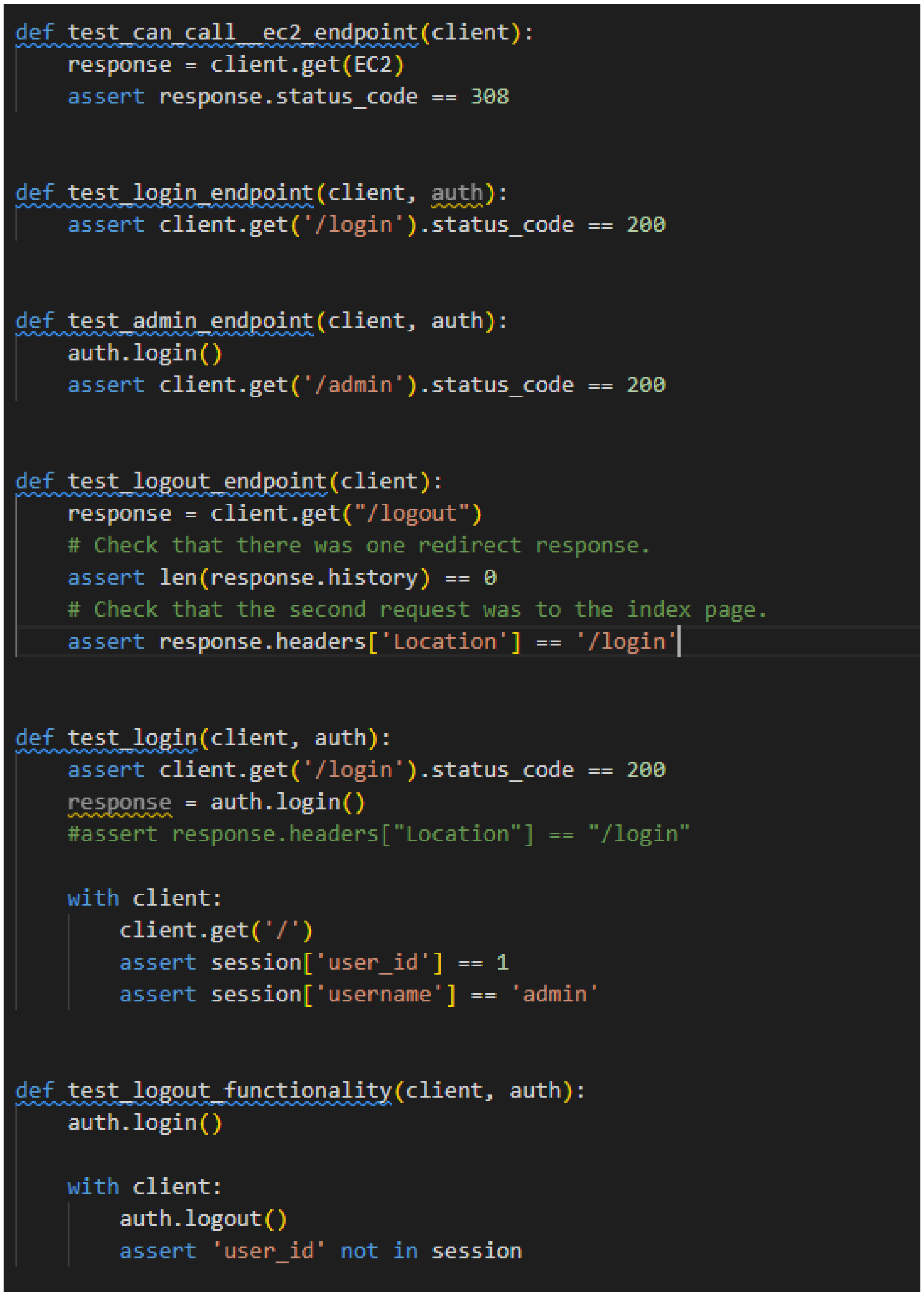
```
import pytest
from website import create_app

# Testing using flask tutorial
# https://flask.palletsprojects.com/en/2.0.x/tutorial/tests/


@pytest.fixture()
def app():
    app = create_app()
    app.config.update({
        "TESTING": True,
    })

    # other setup can go here

    yield app

    # clean up / reset resources here


@pytest.fixture()
def client(app):
    return app.test_client()


@pytest.fixture()
def runner(app):
    return app.test_cli_runner()


class AuthActions(object):
    def __init__(self, client):
        self._client = client

    def login(self, username='dkulis', password='admin'):
        return self._client.post(
            '/login', data={'username': username, 'password': password}
        )

    def logout(self):
        return self._client.get('/logout')


@pytest.fixture()
def auth(client):
    return AuthActions(client)
```

The testing suit is using fixtures to create apps and clients to act as a user. This will allow it to connect to the endpoints and tests if the functionality that user is able to perform, can be used by said user. This is useful for testing authentication when a user tries to access the admin page and they try to input users to login or have them register.

AUTH.PY TESTS

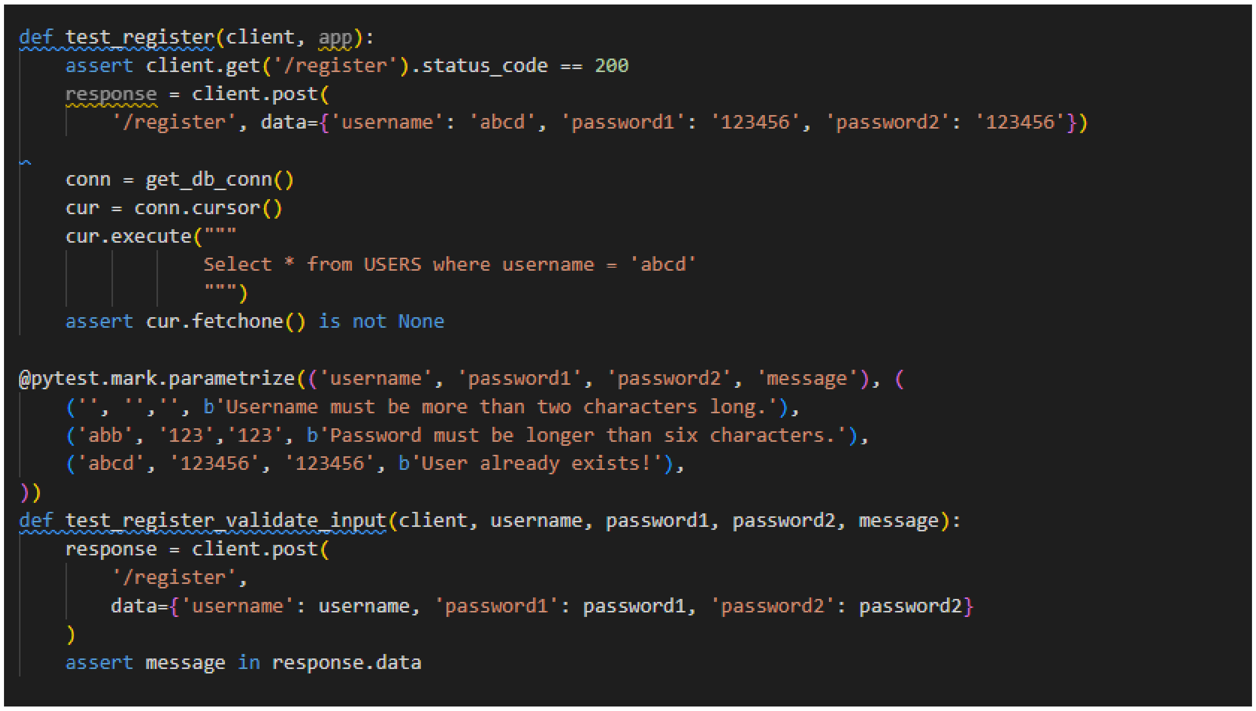


```
platform win32 -- Python 3.10.7, pytest-7.2.0, pluggy-1.0.0 -- C:\Users\Kyurre\AppData\Local\Programs\Python\Pythor
test_auth.py::test_can_call__ec2_endpoint PASSED
test_auth.py::test_login_endpoint PASSED
test_auth.py::test_admin_endpoint PASSED
test_auth.py::test_logout_endpoint PASSED
test_auth.py::test_login PASSED
test_auth.py::test_logout_functionality PASSED
test_auth.py::test_register PASSED
test_auth.py::test_register_validate_input[---Username must be more than two characters long.] PASSED
test_auth.py::test_register_validate_input[abb-123-123-Password must be longer than six characters.] PASSED
test_auth.py::test_register_validate_input[abcd-123456-123456-User already exists!] PASSED
```



```python
def test_can_call__ec2_endpoint(client):
    response = client.get(EC2)
    assert response.status_code == 308


def test_login_endpoint(client, auth):
    assert client.get('/login').status_code == 200


def test_admin_endpoint(client, auth):
    auth.login()
    assert client.get('/admin').status_code == 200


def test_logout_endpoint(client):
    response = client.get("/logout")
    # Check that there was one redirect response.
    assert len(response.history) == 0
    # Check that the second request was to the index page.
    assert response.headers['Location'] == '/login'


def test_login(client, auth):
    assert client.get('/login').status_code == 200
    response = auth.login()
    #assert response.headers["Location"] == "/login"

    with client:
        client.get('/')
        assert session['user_id'] == 1
        assert session['username'] == 'admin'


def test_logout_functionality(client, auth):
    auth.login()

    with client:
        auth.logout()
        assert 'user_id' not in session
```

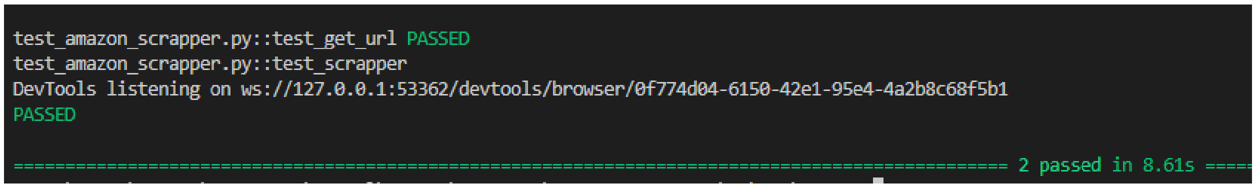
AUTH.PY TESTS CONT.



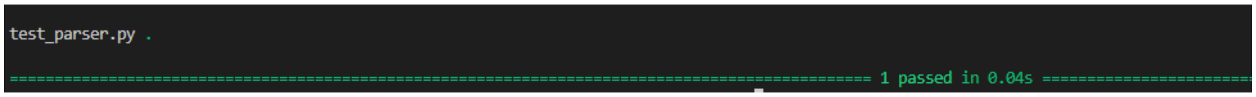
```
def test_register(client, app):
    assert client.get('/register').status_code == 200
    response = client.post(
        '/register', data={'username': 'abcd', 'password1': '123456', 'password2': '123456'})

    conn = get_db_conn()
    cur = conn.cursor()
    cur.execute("""
                Select * from USERS where username = 'abcd'
                """)
    assert cur.fetchone() is not None

@pytest.mark.parametrize(('username', 'password1', 'password2', 'message'), (
    ('', '','', b'Username must be more than two characters long.'),
    ('abb', '123','123', b'Password must be longer than six characters.'),
    ('abcd', '123456', '123456', b'User already exists!'),
))
def test_register_validate_input(client, username, password1, password2, message):
    response = client.post(
        '/register',
        data={'username': username, 'password1': password1, 'password2': password2}
    )
    assert message in response.data
```

A decision was made to test these specific endpoints because they were the first step into the GPYou website. The login page is to function as intended and lets a user log in as the correct user. Going along that thought process, to make sure that when you register as a user, one maybe prompted with correct errors if they misinput and are told if their username already exists.

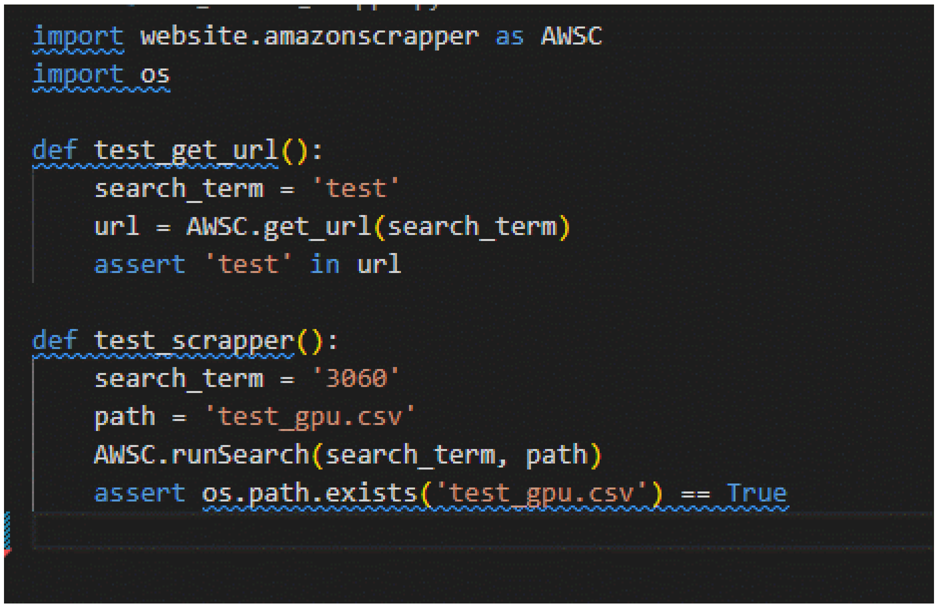
SCRAPPER AND PARSER



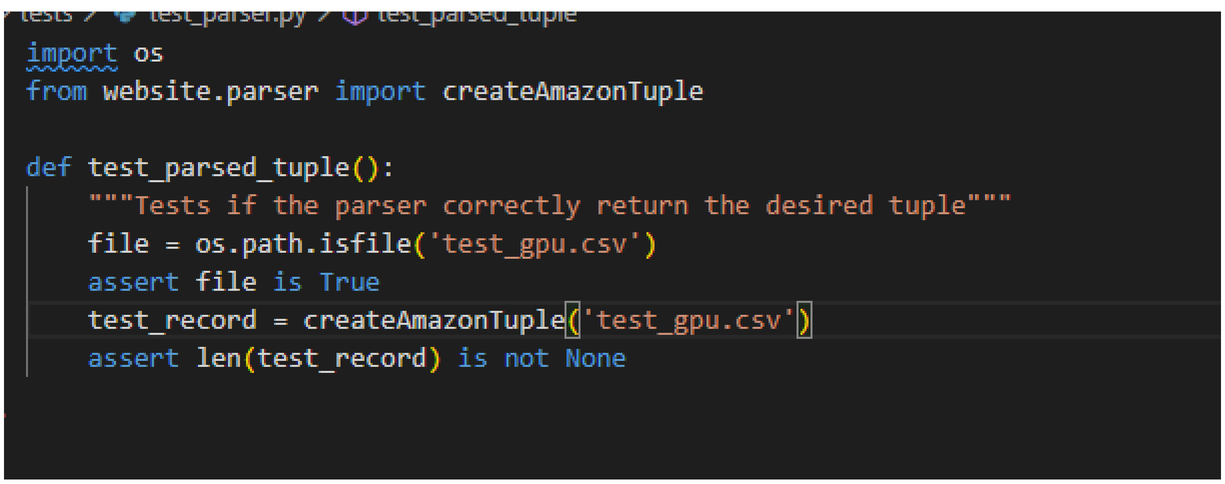
```
test_amazon_scrapper.py::test_get_url PASSED
test_amazon_scrapper.py::test_scrapper
DevTools listening on ws://127.0.0.1:53362/devtools/browser/0f774d04-6150-42e1-95e4-4a2b8c68f5b1
PASSED

========================================================================================== 2 passed in 8.61s =====
```



```
test_parser.py .

=============================================================================== 1 passed in 0.04s =====================
```



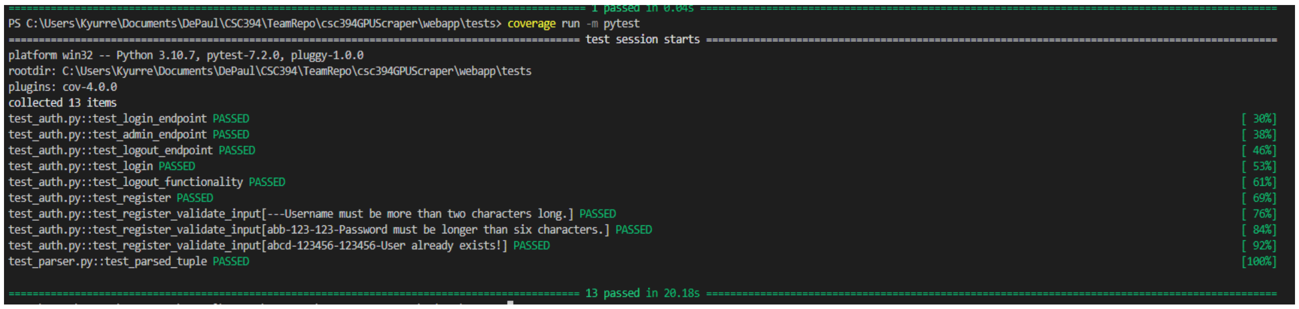
```python
import website.amazonscrapper as AWSC
import os

def test_get_url():
    search_term = 'test'
    url = AWSC.get_url(search_term)
    assert 'test' in url

def test_scrapper():
    search_term = '3060'
    path = 'test_gpu.csv'
    AWSC.runSearch(search_term, path)
    assert os.path.exists('test_gpu.csv') == True
```



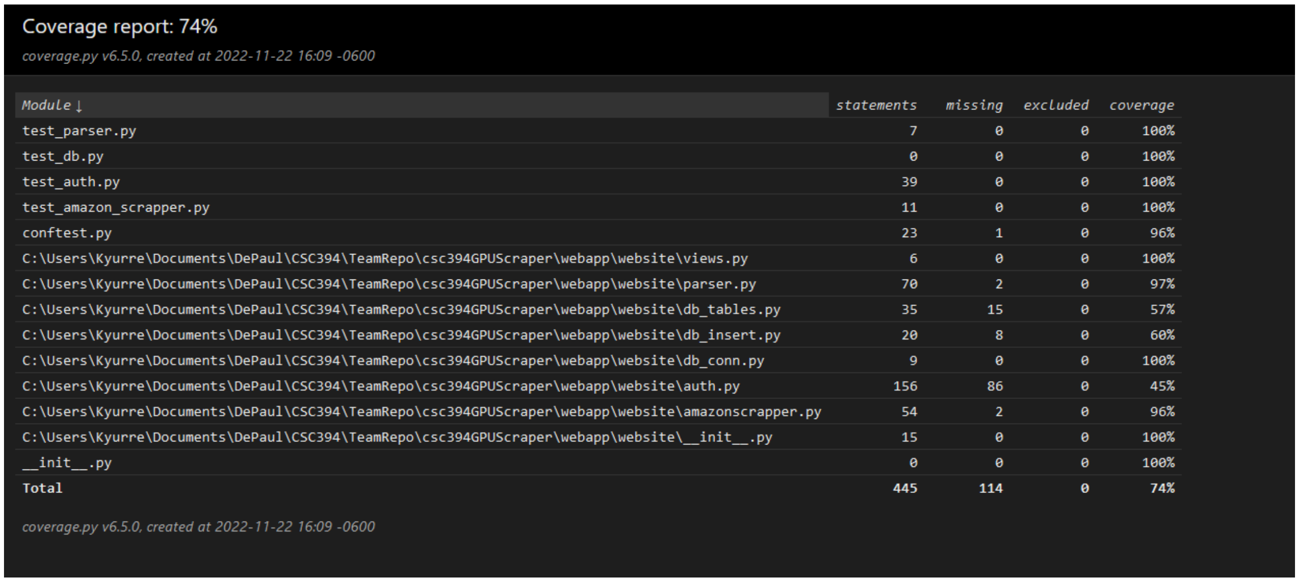
```python
import os
from website.parser import createAmazonTuple

def test_parsed_tuple():
    """Tests if the parser correctly return the desired tuple"""
    file = os.path.isfile('test_gpu.csv')
    assert file is True
    test_record = createAmazonTuple('test_gpu.csv')
    assert len(test_record) is not None
```

Since a lot of the parser methods were helper methods, its required to validate if a proper tuple was created. A relevant search of an existing GPU such as 3060 helped ensure that the parser was creating tuples and was able to access the csv created by the scrapper.

FINAL TEST COVERAGE REPORT



```
========================================================================================== 1 passed in 0.04s ==========================================================================================
PS C:\Users\Kyurre\Documents\DePaul\CSC394\TeamRepo\csc394GPUScraper\webapp\tests> coverage run -m pytest
========================================================================================= test session starts =========================================================================================
platform win32 -- Python 3.10.7, pytest-7.2.0, pluggy-1.0.0
rootdir: C:\Users\Kyurre\Documents\DePaul\CSC394\TeamRepo\csc394GPUScraper\webapp\tests
plugins: cov-4.0.0
collected 13 items
test_auth.py::test_login_endpoint PASSED                                                                                                                                                      [ 30%]
test_auth.py::test_admin_endpoint PASSED                                                                                                                                                      [ 38%]
test_auth.py::test_logout_endpoint PASSED                                                                                                                                                     [ 46%]
test_auth.py::test_login PASSED                                                                                                                                                               [ 53%]
test_auth.py::test_logout_functionality PASSED                                                                                                                                                [ 61%]
test_auth.py::test_register PASSED                                                                                                                                                            [ 69%]
test_auth.py::test_register_validate_input[---Username must be more than two characters long.] PASSED                                                                                         [ 76%]
test_auth.py::test_register_validate_input[abb-123-123-Password must be longer than six characters.] PASSED                                                                                   [ 84%]
test_auth.py::test_register_validate_input[abcd-123456-123456-User already exists!] PASSED                                                                                                    [ 92%]
test_parser.py::test_parsed_tuple PASSED                                                                                                                                                      [100%]

========================================================================================= 13 passed in 20.18s =========================================================================================
```



Coverage report: 74%

coverage.py v6.5.0, created at 2022-11-22 16:09 -0600

Module ↓	*statements*	*missing*	*excluded*	*coverage*
test_parser.py	7	0	0	100%
test_db.py	0	0	0	100%
test_auth.py	39	0	0	100%
test_amazon_scrapper.py	11	0	0	100%
conftest.py	23	1	0	96%
C:\Users\Kyurre\Documents\DePaul\CSC394\TeamRepo\csc394GPUScraper\webapp\website\views.py	6	0	0	100%
C:\Users\Kyurre\Documents\DePaul\CSC394\TeamRepo\csc394GPUScraper\webapp\website\parser.py	70	2	0	97%
C:\Users\Kyurre\Documents\DePaul\CSC394\TeamRepo\csc394GPUScraper\webapp\website\db_tables.py	35	15	0	57%
C:\Users\Kyurre\Documents\DePaul\CSC394\TeamRepo\csc394GPUScraper\webapp\website\db_insert.py	20	8	0	60%
C:\Users\Kyurre\Documents\DePaul\CSC394\TeamRepo\csc394GPUScraper\webapp\website\db_conn.py	9	0	0	100%
C:\Users\Kyurre\Documents\DePaul\CSC394\TeamRepo\csc394GPUScraper\webapp\website\auth.py	156	86	0	45%
C:\Users\Kyurre\Documents\DePaul\CSC394\TeamRepo\csc394GPUScraper\webapp\website\amazonscrapper.py	54	2	0	96%
C:\Users\Kyurre\Documents\DePaul\CSC394\TeamRepo\csc394GPUScraper\webapp\website\__init__.py	15	0	0	100%
__init__.py	0	0	0	100%
Total	**445**	**114**	**0**	**74%**

coverage.py v6.5.0, created at 2022-11-22 16:09 -0600

DEMO

Unfortunately, our team wasn't able to create Demo Scripts or proper walkthrough during the majority of the course. We were able to present a script for the final demo which is included below.

Final Demo script:
Admin Panel:

- Show login as user vs administrator
  - Navbar should be different for both
- Deleting users as admin

Search:

- Searching for GPU with things like 2080
  - Visit link to show it is the same as scrapped
- Searching for things like
  - Asus Laptop and it shouldn't populate table

POC/Demo1:
Admin Panel:

- Login as admin
- Remove user
- Change user password
- Change admin privileges
- Crate admin user

Demo2:
Admin Panel:

- Show the prior functionality
- Show the progress for the scrapper

Demo3:
Admin Panel:

- Prior Functionality

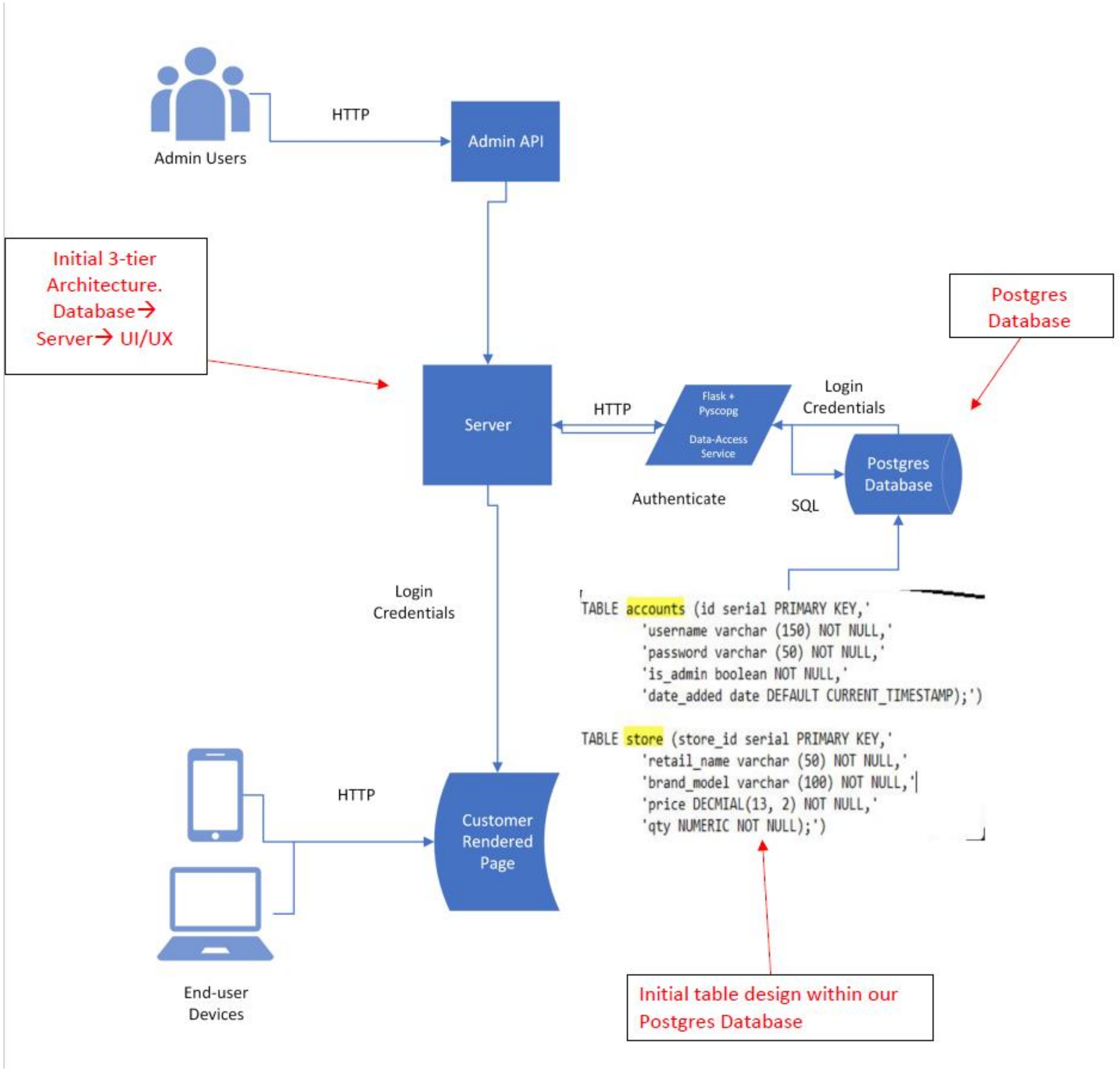
Scrapper and Parser:

- Show the parsed data inside the database
- Show the creation of csv from the scrapper

DATA LAYER

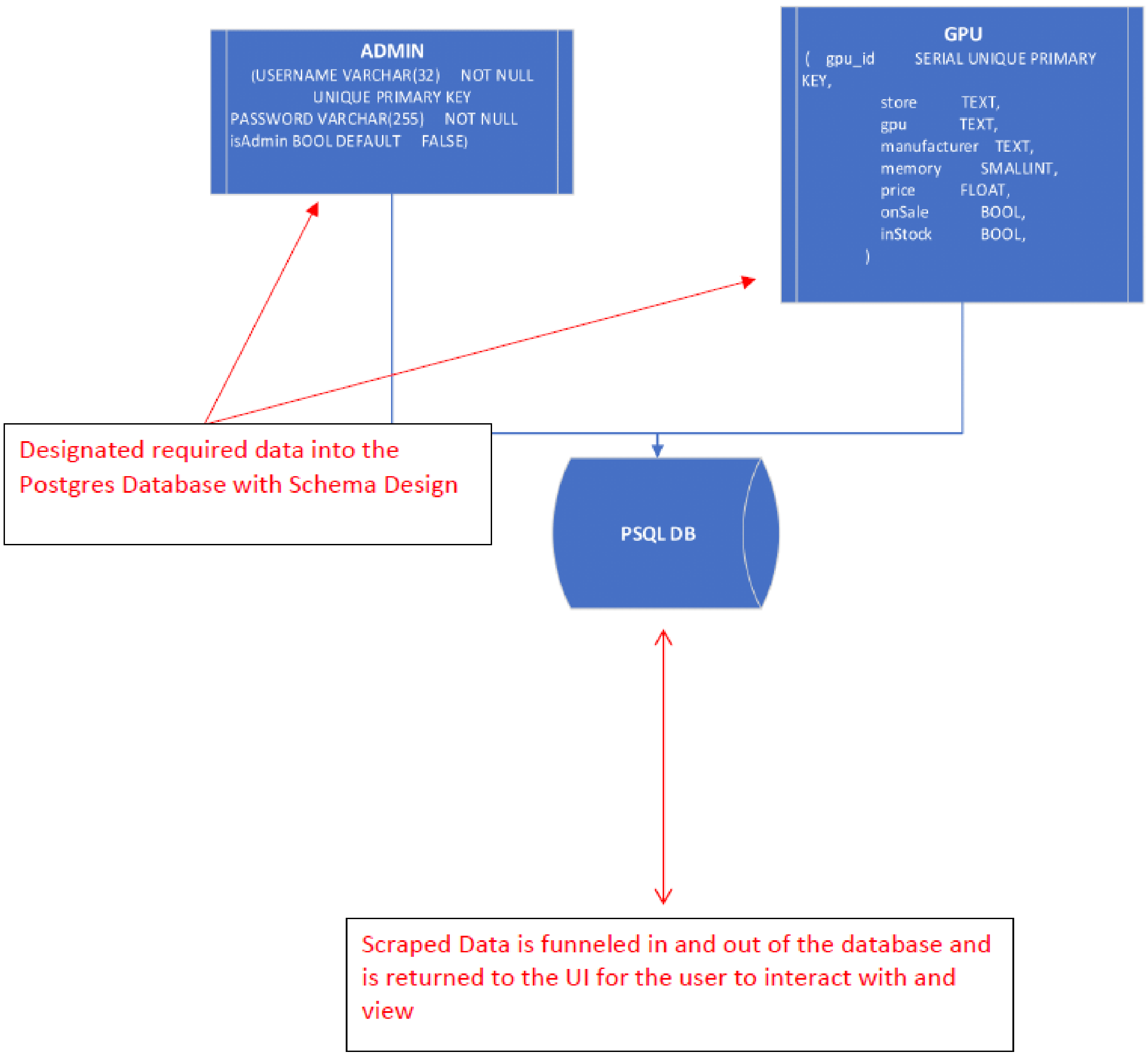
This section will highlight the overall layers of the initial software architecture from database to UI, as well as target the admin perspective for the admin panel.

DEMO 1 MODEL

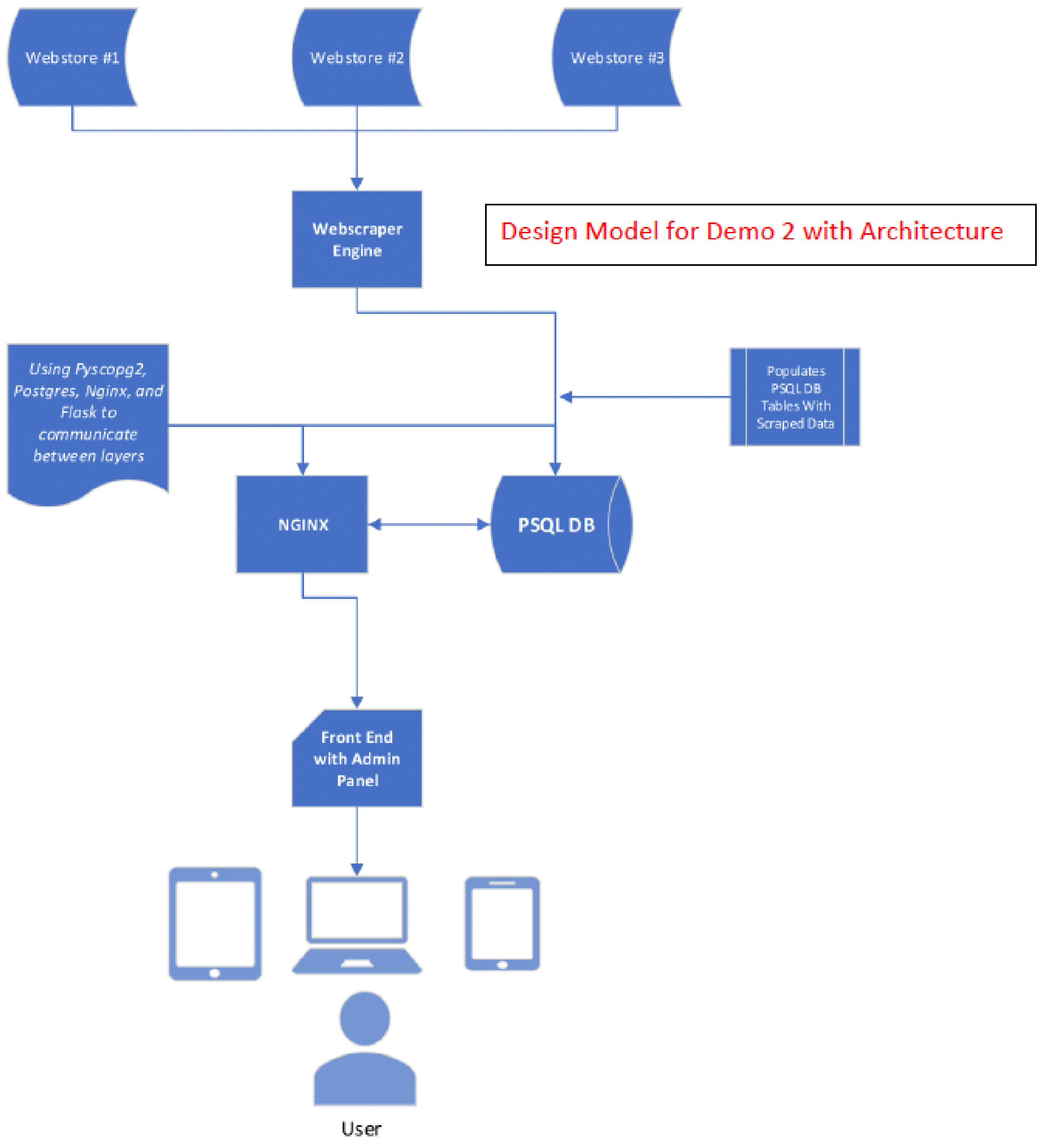


```
TABLE accounts (id serial PRIMARY KEY,'
        'username varchar (150) NOT NULL,'
        'password varchar (50) NOT NULL,'
        'is_admin boolean NOT NULL,'
        'date_added date DEFAULT CURRENT_TIMESTAMP);')

TABLE store (store_id serial PRIMARY KEY,'
        'retail_name varchar (50) NOT NULL,'
        'brand_model varchar (100) NOT NULL,'
        'price DECMIAL(13, 2) NOT NULL,'
        'qty NUMERIC NOT NULL);')
```

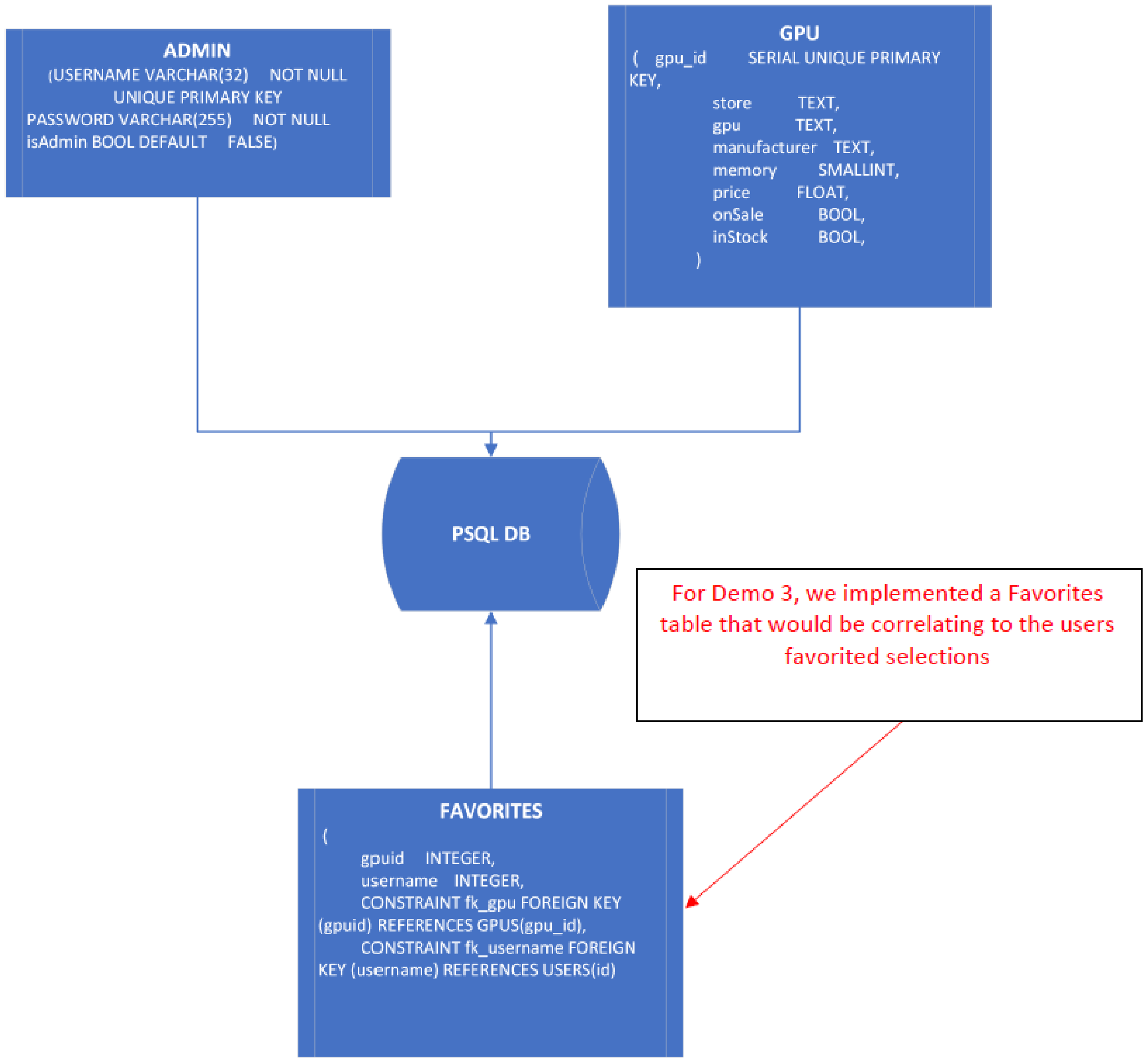
DEMO 2 DATA BASE SCHEMA


ADMIN
(USERNAME VARCHAR(32) NOT NULL UNIQUE PRIMARY KEY
PASSWORD VARCHAR(255) NOT NULL
isAdmin BOOL DEFAULT FALSE)
GPU
(gpu_id SERIAL UNIQUE PRIMARY KEY,
store TEXT,
gpu TEXT,
manufacturer TEXT,
memory SMALLINT,
price FLOAT,
onSale BOOL,
inStock BOOL,
)
PSQL DB
Designated required data into the Postgres Database with Schema Design
Scraped Data is funneled in and out of the database and is returned to the UI for the user to interact with and view

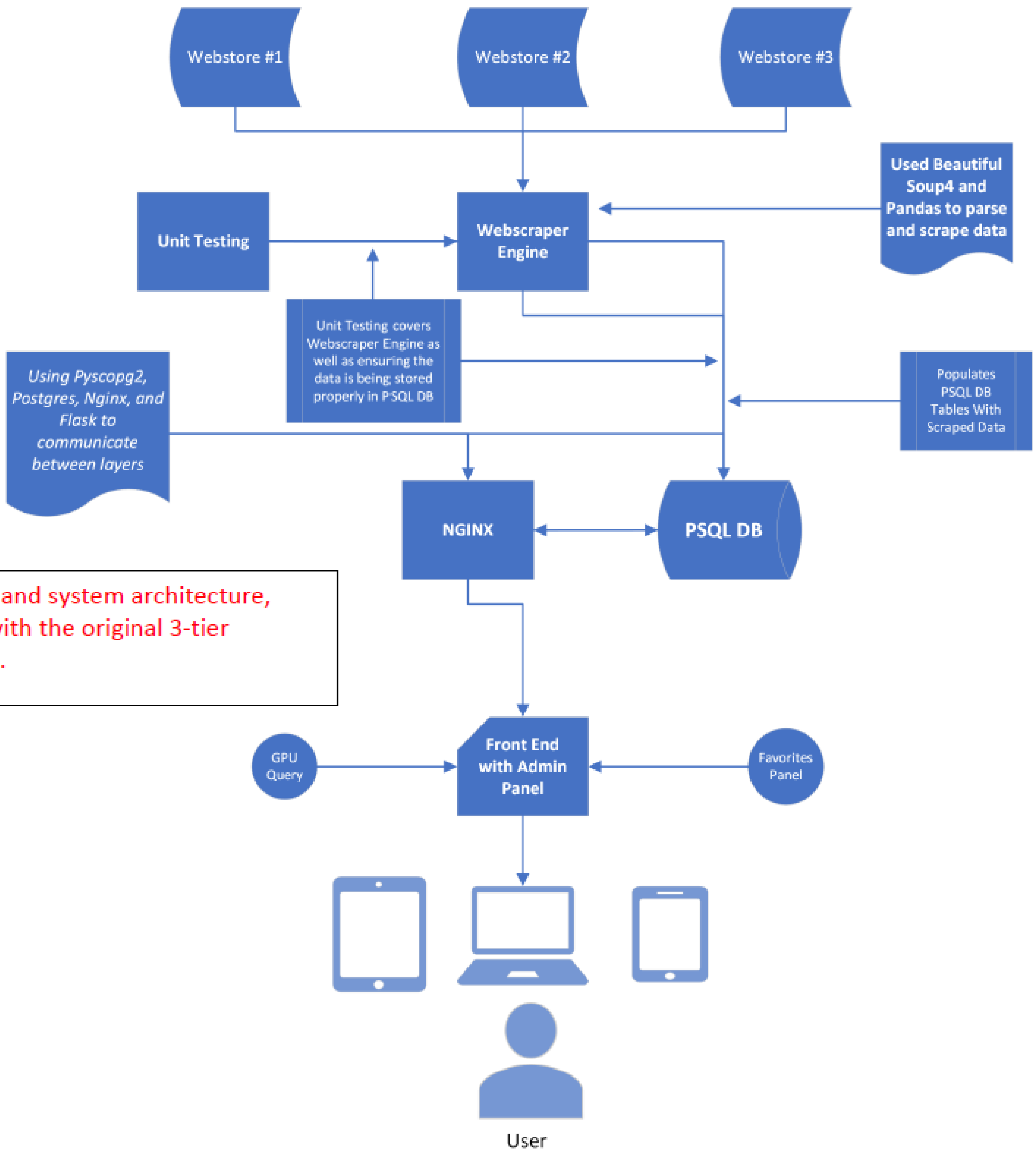
DEMO 2 SYSTEM ARCHITECTURE


Webstore #1
Webstore #2
Webstore #3
Webscraper Engine
Design Model for Demo 2 with Architecture
Using Pyscopg2, Postgres, Nginx, and Flask to communicate between layers
Populates PSQL DB Tables With Scraped Data
NGINX
PSQL DB
Front End with Admin Panel
User

DEMO 3 DATABASE SCHEMA

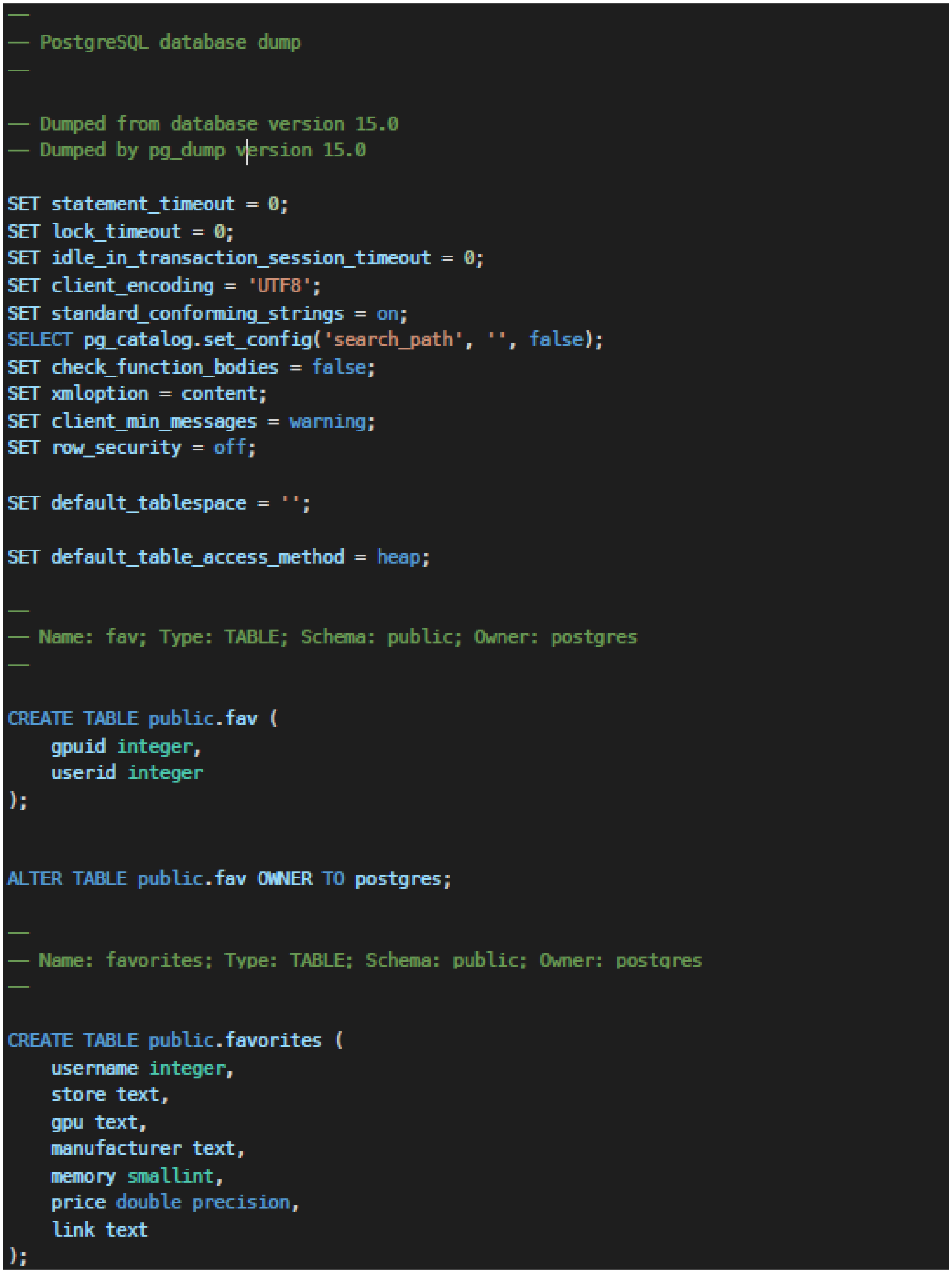

ADMIN
(USERNAME VARCHAR(32) NOT NULL UNIQUE PRIMARY KEY
PASSWORD VARCHAR(255) NOT NULL
isAdmin BOOL DEFAULT FALSE)
GPU
(gpu_id SERIAL UNIQUE PRIMARY KEY,
store TEXT,
gpu TEXT,
manufacturer TEXT,
memory SMALLINT,
price FLOAT,
onSale BOOL,
inStock BOOL,
)
PSQL DB
For Demo 3, we implemented a Favorites table that would be correlating to the users favorited selections
FAVORITES
(
gpuid INTEGER,
username INTEGER,
CONSTRAINT fk_gpu FOREIGN KEY (gpuid) REFERENCES GPUS(gpu_id),
CONSTRAINT fk_username FOREIGN KEY (username) REFERENCES USERS(id)

DEMO 3 SYSTEM ARCHITECTURE

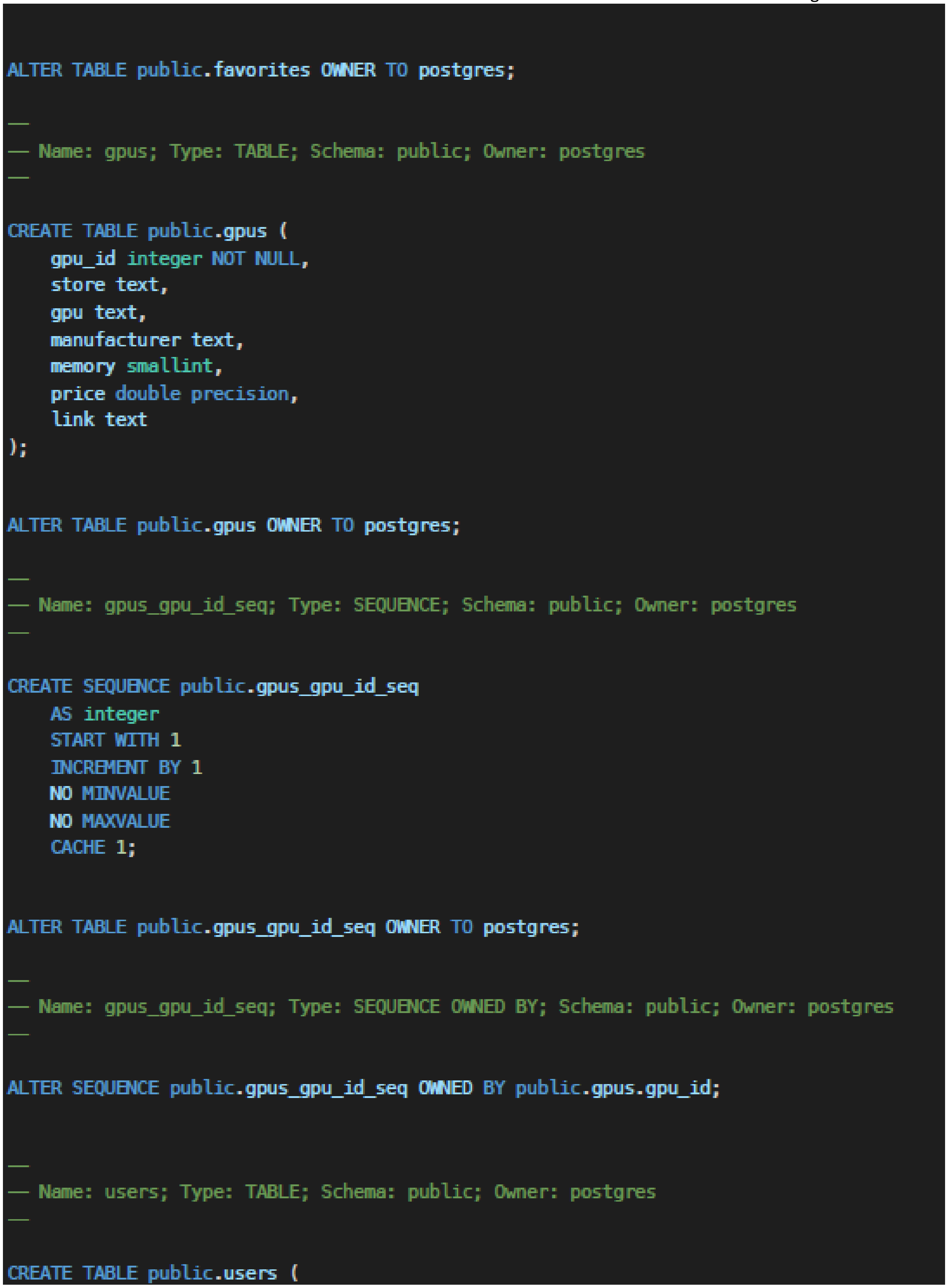

Webstore #1
Webstore #2
Webstore #3
Used Beautiful Soup4 and Pandas to parse and scrape data
Unit Testing
Webscraper Engine
Unit Testing covers Webscraper Engine as well as ensuring the data is being stored properly in PSQL DB
Using Pyscopg2, Postgres, Nginx, and Flask to communicate between layers
Populates PSQL DB Tables With Scraped Data
NGINX
PSQL DB
GPU Query
Front End with Admin Panel
Favorites Panel
User

Final design and system architecture, consistent with the original 3-tier architecture.

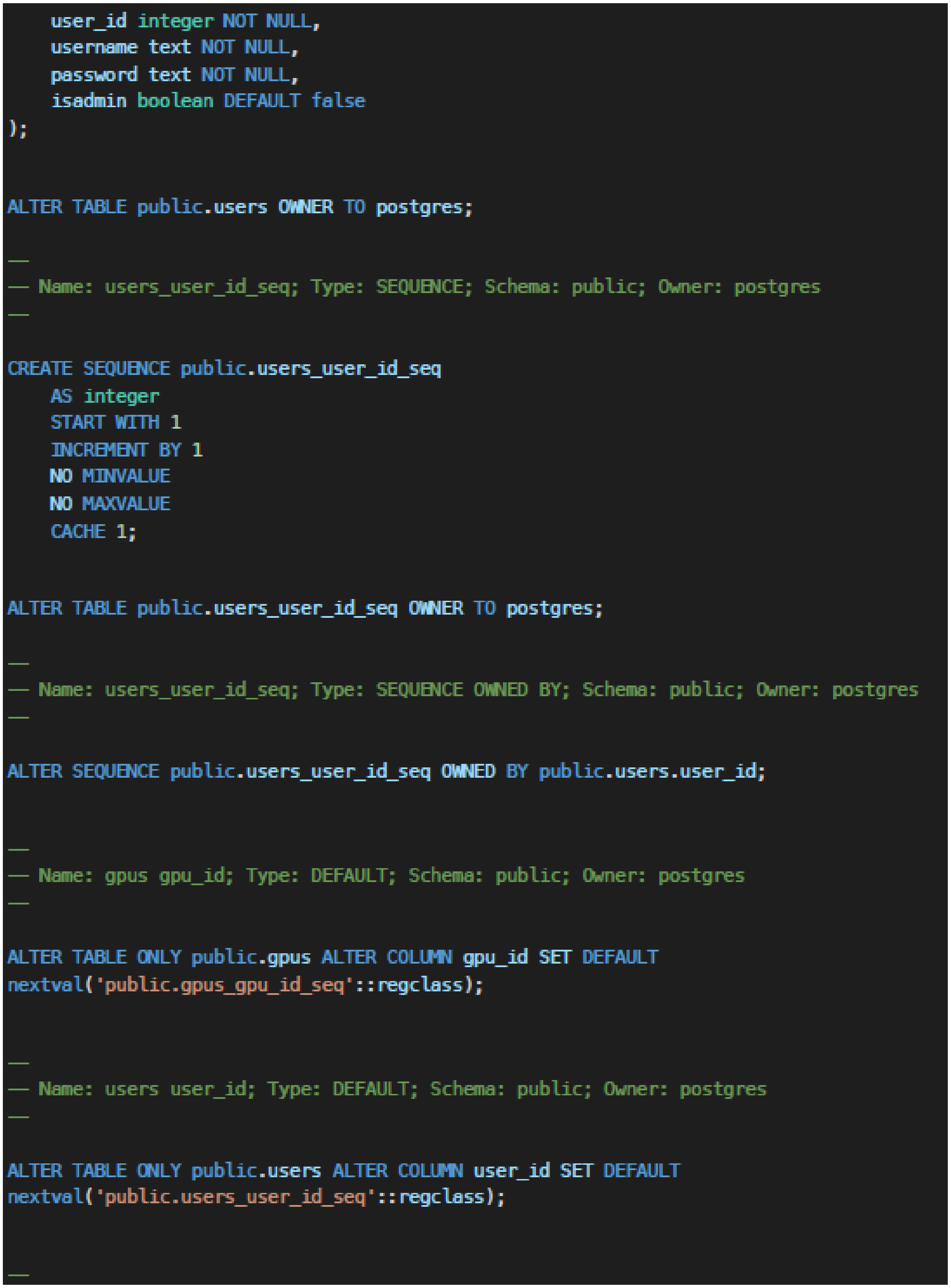
DATABASE SQL CODE AND PG DUMP TO CREATE SCHEMA FOR GPYOU SCRAPER



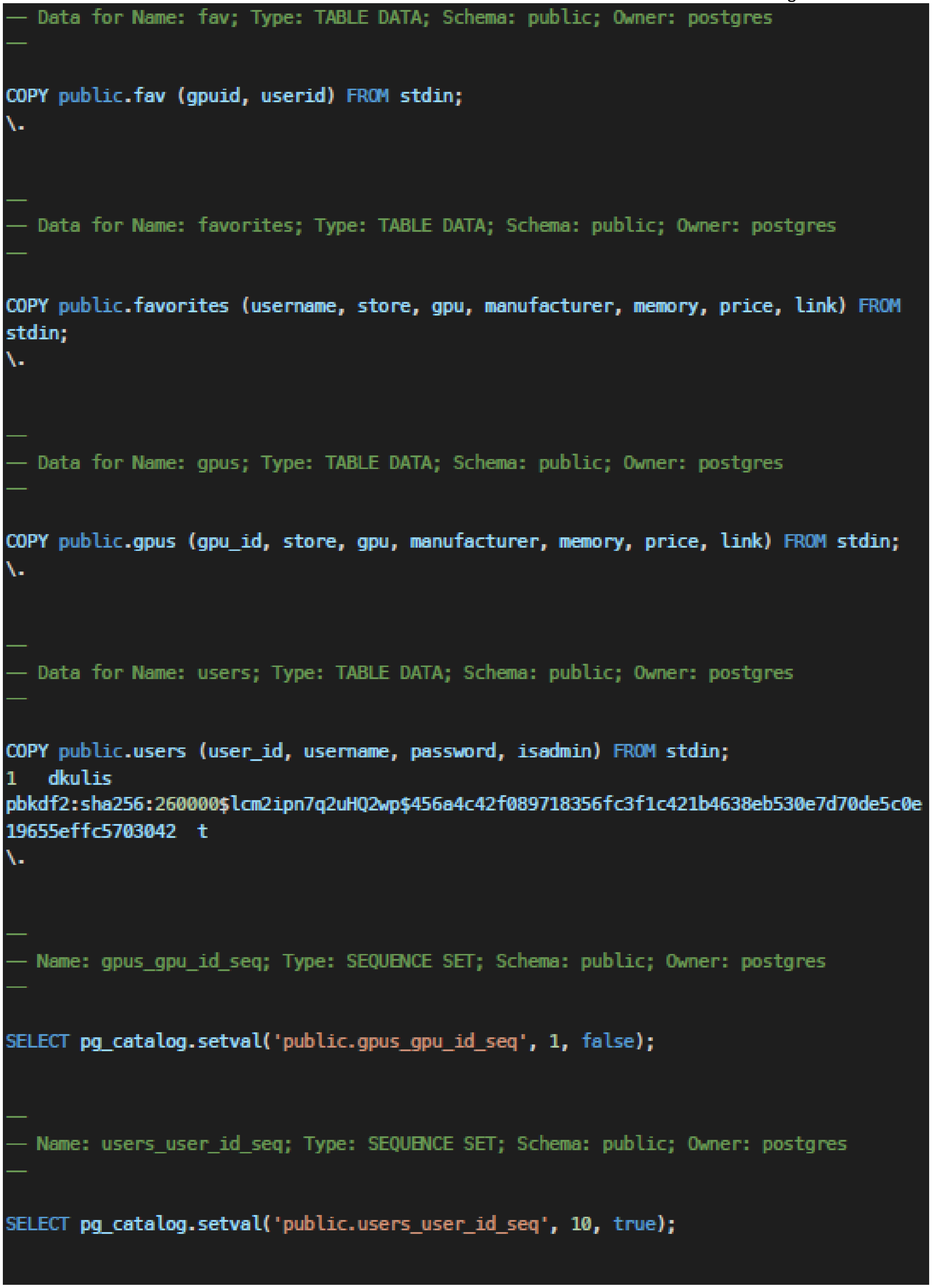
```
--
-- PostgreSQL database dump
--

-- Dumped from database version 15.0
-- Dumped by pg_dump version 15.0

SET statement_timeout = 0;
SET lock_timeout = 0;
SET idle_in_transaction_session_timeout = 0;
SET client_encoding = 'UTF8';
SET standard_conforming_strings = on;
SELECT pg_catalog.set_config('search_path', '', false);
SET check_function_bodies = false;
SET xmloption = content;
SET client_min_messages = warning;
SET row_security = off;

SET default_tablespace = '';

SET default_table_access_method = heap;

--
-- Name: fav; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public.fav (
    gpuid integer,
    userid integer
);


ALTER TABLE public.fav OWNER TO postgres;

--
-- Name: favorites; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public.favorites (
    username integer,
    store text,
    gpu text,
    manufacturer text,
    memory smallint,
    price double precision,
    link text
);
```



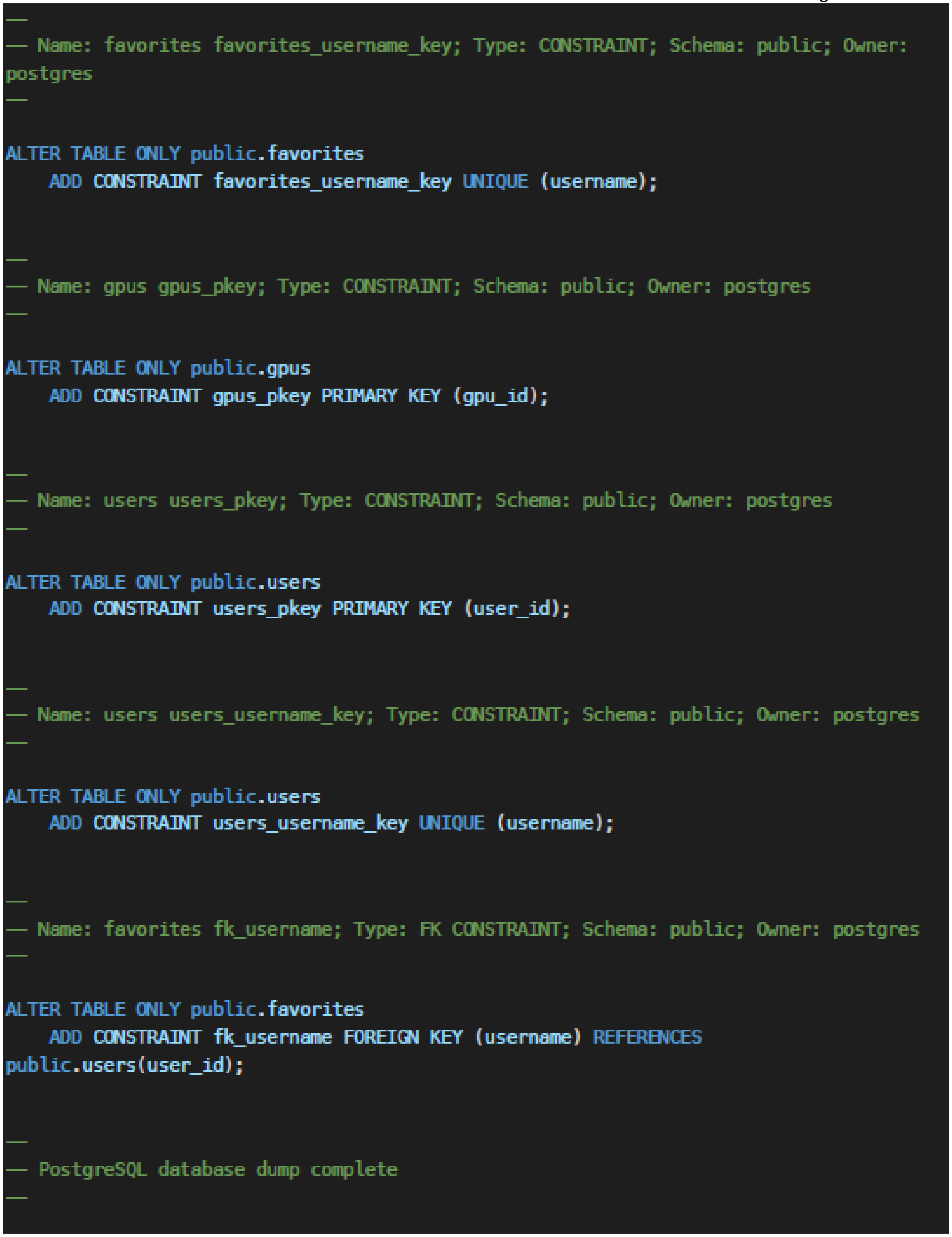
```
ALTER TABLE public.favorites OWNER TO postgres;

--
-- Name: gpus; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public.gpus (
    gpu_id integer NOT NULL,
    store text,
    gpu text,
    manufacturer text,
    memory smallint,
    price double precision,
    link text
);


ALTER TABLE public.gpus OWNER TO postgres;

--
-- Name: gpus_gpu_id_seq; Type: SEQUENCE; Schema: public; Owner: postgres
--

CREATE SEQUENCE public.gpus_gpu_id_seq
    AS integer
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;


ALTER TABLE public.gpus_gpu_id_seq OWNER TO postgres;

--
-- Name: gpus_gpu_id_seq; Type: SEQUENCE OWNED BY; Schema: public; Owner: postgres
--

ALTER SEQUENCE public.gpus_gpu_id_seq OWNED BY public.gpus.gpu_id;


--
-- Name: users; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public.users (
```
]



```
    user_id integer NOT NULL,
    username text NOT NULL,
    password text NOT NULL,
    isadmin boolean DEFAULT false
);


ALTER TABLE public.users OWNER TO postgres;

--
-- Name: users_user_id_seq; Type: SEQUENCE; Schema: public; Owner: postgres
--

CREATE SEQUENCE public.users_user_id_seq
    AS integer
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;


ALTER TABLE public.users_user_id_seq OWNER TO postgres;

--
-- Name: users_user_id_seq; Type: SEQUENCE OWNED BY; Schema: public; Owner: postgres
--

ALTER SEQUENCE public.users_user_id_seq OWNED BY public.users.user_id;


--
-- Name: gpus gpu_id; Type: DEFAULT; Schema: public; Owner: postgres
--

ALTER TABLE ONLY public.gpus ALTER COLUMN gpu_id SET DEFAULT
nextval('public.gpus_gpu_id_seq'::regclass);


--
-- Name: users user_id; Type: DEFAULT; Schema: public; Owner: postgres
--

ALTER TABLE ONLY public.users ALTER COLUMN user_id SET DEFAULT
nextval('public.users_user_id_seq'::regclass);


--
```



```
-- Data for Name: fav; Type: TABLE DATA; Schema: public; Owner: postgres
--

COPY public.fav (gpuid, userid) FROM stdin;
\.


--
-- Data for Name: favorites; Type: TABLE DATA; Schema: public; Owner: postgres
--

COPY public.favorites (username, store, gpu, manufacturer, memory, price, link) FROM
stdin;
\.


--
-- Data for Name: gpus; Type: TABLE DATA; Schema: public; Owner: postgres
--

COPY public.gpus (gpu_id, store, gpu, manufacturer, memory, price, link) FROM stdin;
\.


--
-- Data for Name: users; Type: TABLE DATA; Schema: public; Owner: postgres
--

COPY public.users (user_id, username, password, isadmin) FROM stdin;
1	dkulis
pbkdf2:sha256:260000$lcm2ipn7q2uHQ2wp$456a4c42f089718356fc3f1c421b4638eb530e7d70de5c0e
19655effc5703042	t
\.


--
-- Name: gpus_gpu_id_seq; Type: SEQUENCE SET; Schema: public; Owner: postgres
--

SELECT pg_catalog.setval('public.gpus_gpu_id_seq', 1, false);


--
-- Name: users_user_id_seq; Type: SEQUENCE SET; Schema: public; Owner: postgres
--

SELECT pg_catalog.setval('public.users_user_id_seq', 10, true);
```



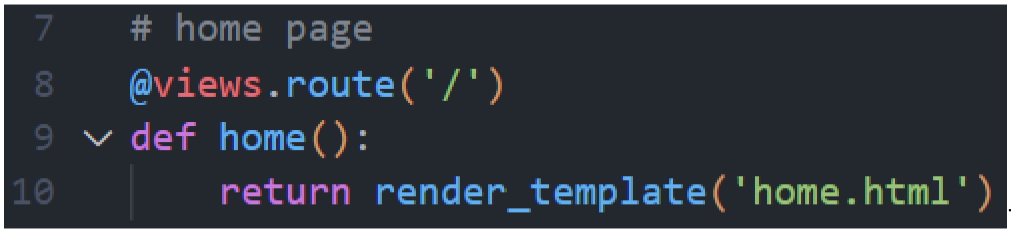
```sql
--
-- Name: favorites favorites_username_key; Type: CONSTRAINT; Schema: public; Owner:
postgres
--

ALTER TABLE ONLY public.favorites
    ADD CONSTRAINT favorites_username_key UNIQUE (username);


--
-- Name: gpus gpus_pkey; Type: CONSTRAINT; Schema: public; Owner: postgres
--

ALTER TABLE ONLY public.gpus
    ADD CONSTRAINT gpus_pkey PRIMARY KEY (gpu_id);


--
-- Name: users users_pkey; Type: CONSTRAINT; Schema: public; Owner: postgres
--

ALTER TABLE ONLY public.users
    ADD CONSTRAINT users_pkey PRIMARY KEY (user_id);


--
-- Name: users users_username_key; Type: CONSTRAINT; Schema: public; Owner: postgres
--

ALTER TABLE ONLY public.users
    ADD CONSTRAINT users_username_key UNIQUE (username);


--
-- Name: favorites fk_username; Type: FK CONSTRAINT; Schema: public; Owner: postgres
--

ALTER TABLE ONLY public.favorites
    ADD CONSTRAINT fk_username FOREIGN KEY (username) REFERENCES
public.users(user_id);


--
-- PostgreSQL database dump complete
--
```

LOGIC LAYER

VIEWS.PY

- views.py contains routes that do not make any POST requests.

Home



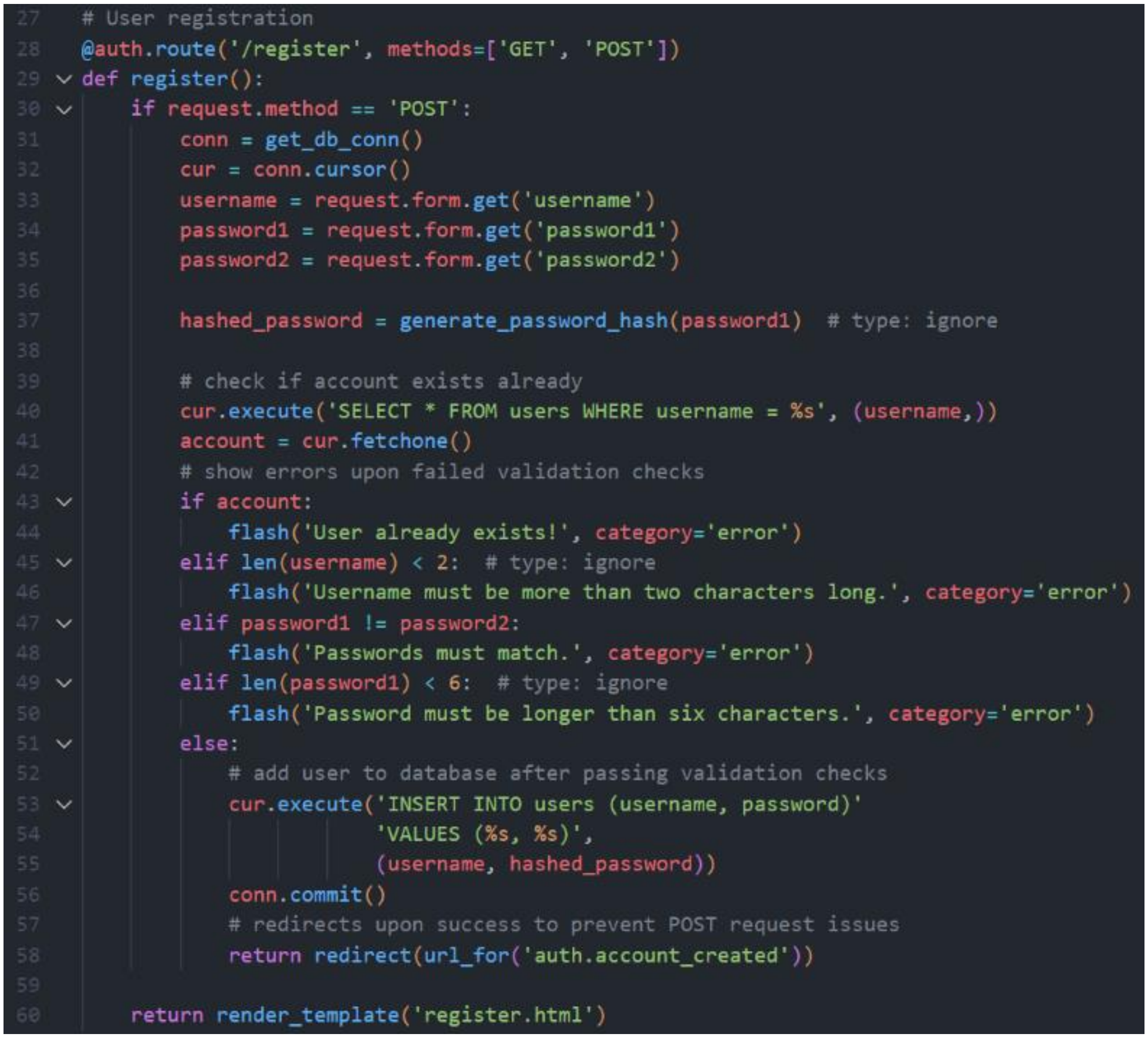
```
# home page
@views.route('/')
def home():
    return render_template('home.html')
```

The only route present in views.py, '/' is used specifically for the home page. It is kept separate from all routes with functional properties.

AUTH.PY

- auth.py contains all routes that handle GET and POST requests. It is responsible for the majority of the websites functionality.

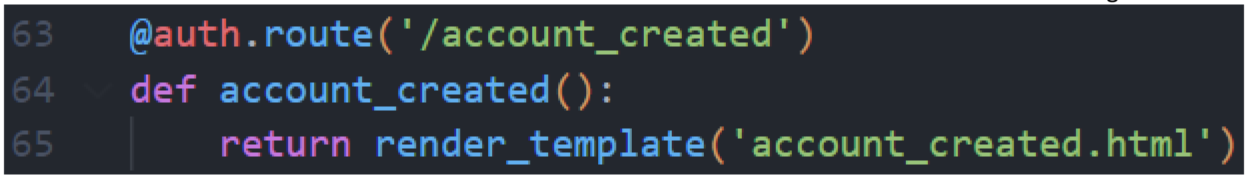
REGISTRATION



```
# User registration
@auth.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        conn = get_db_conn()
        cur = conn.cursor()
        username = request.form.get('username')
        password1 = request.form.get('password1')
        password2 = request.form.get('password2')

        hashed_password = generate_password_hash(password1)  # type: ignore

        # check if account exists already
        cur.execute('SELECT * FROM users WHERE username = %s', (username,))
        account = cur.fetchone()
        # show errors upon failed validation checks
        if account:
            flash('User already exists!', category='error')
        elif len(username) < 2:  # type: ignore
            flash('Username must be more than two characters long.', category='error')
        elif password1 != password2:
            flash('Passwords must match.', category='error')
        elif len(password1) < 6:  # type: ignore
            flash('Password must be longer than six characters.', category='error')
        else:
            # add user to database after passing validation checks
            cur.execute('INSERT INTO users (username, password)'
                        'VALUES (%s, %s)',
                        (username, hashed_password))
            conn.commit()
            # redirects upon success to prevent POST request issues
            return redirect(url_for('auth.account_created'))

    return render_template('register.html')
```

The purpose of the '/register' route is user registration. The route handles both GET and POST requests to receive data from the user and then insert the data into the USERS table within the database.

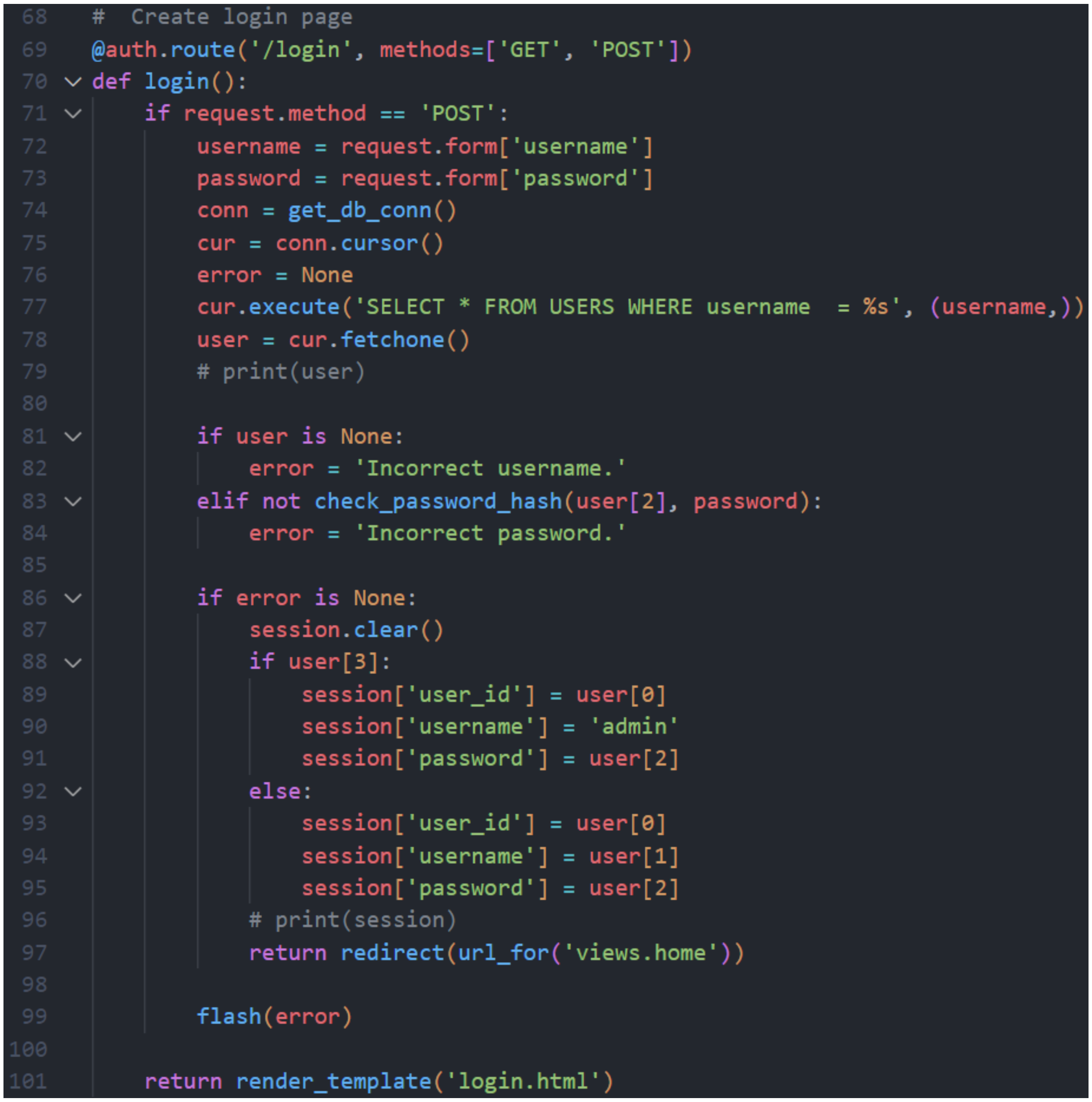
The associated register() function assigns values submitted from the registration form into username, password1, and password2. The password is secured using werkzeug hashes. These values are tested to meet certain requirements before the user is successfully registered. Upon successful registration, the user is redirected to route '/account_created' to avoid duplicate POST requests from staying on the same page.



```python
@auth.route('/account_created')
def account_created():
    return render_template('account_created.html')
```

‘/account_created’ prevents duplicate POST requests that occur upon registration. It is not in views.py because it is used for redirection for a route within auth.py.

LOGIN/LOGOUT

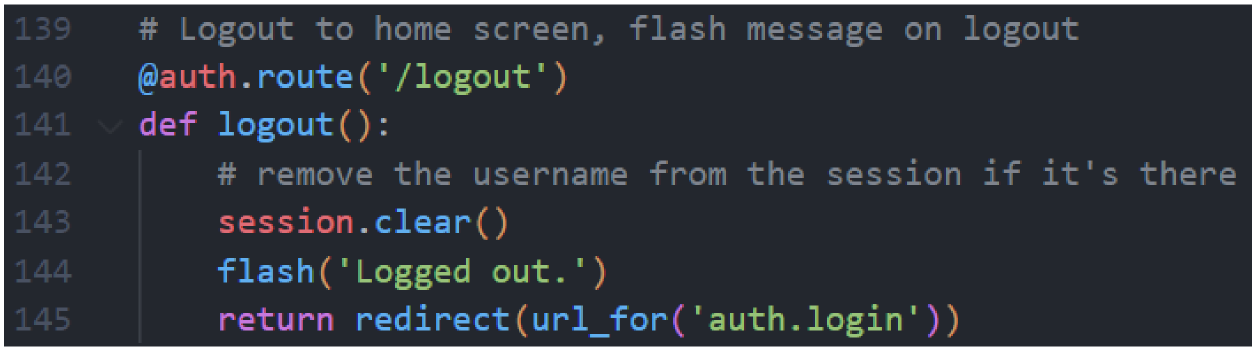


```
#  Create login page
@auth.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        conn = get_db_conn()
        cur = conn.cursor()
        error = None
        cur.execute('SELECT * FROM USERS WHERE username  = %s', (username,))
        user = cur.fetchone()
        # print(user)

        if user is None:
            error = 'Incorrect username.'
        elif not check_password_hash(user[2], password):
            error = 'Incorrect password.'

        if error is None:
            session.clear()
            if user[3]:
                session['user_id'] = user[0]
                session['username'] = 'admin'
                session['password'] = user[2]
            else:
                session['user_id'] = user[0]
                session['username'] = user[1]
                session['password'] = user[2]
            # print(session)
            return redirect(url_for('views.home'))

        flash(error)

    return render_template('login.html')
```

The ‘/login’ route allows a user to log into the website for increased user functionality. The route handles GET and POST requests. The GET method requests the username and the password from the associated login form. The POST method stores user information within a Flask Session.

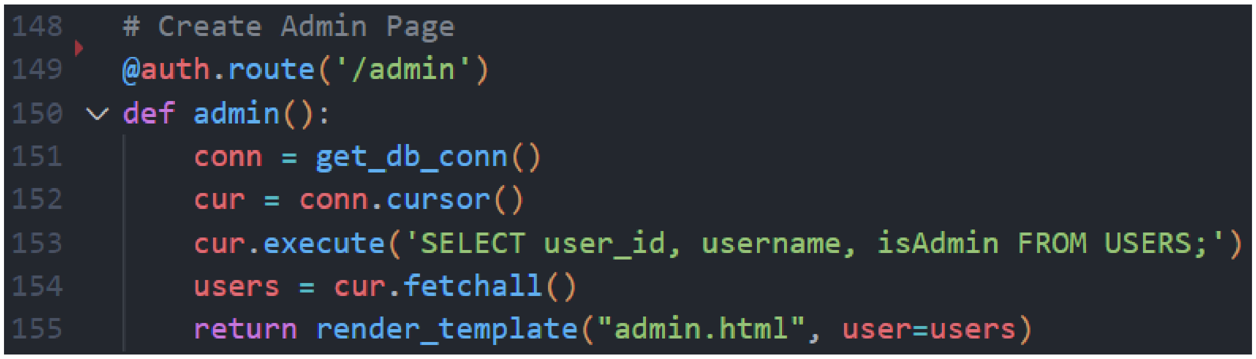
Upon submission of the login form, the login() function checks whether the username belongs to an associated account and if the provided password matches the hashed password stored in the database.

If a username and password combination exists in the USERS table, a session is created. The type of session depends on whether the user is an administrator or not. An administrator will be assigned a session that allows access to the admin panel on the navbar.



```python
# Logout to home screen, flash message on logout
@auth.route('/logout')
def logout():
    # remove the username from the session if it's there
    session.clear()
    flash('Logged out.')
    return redirect(url_for('auth.login'))
```

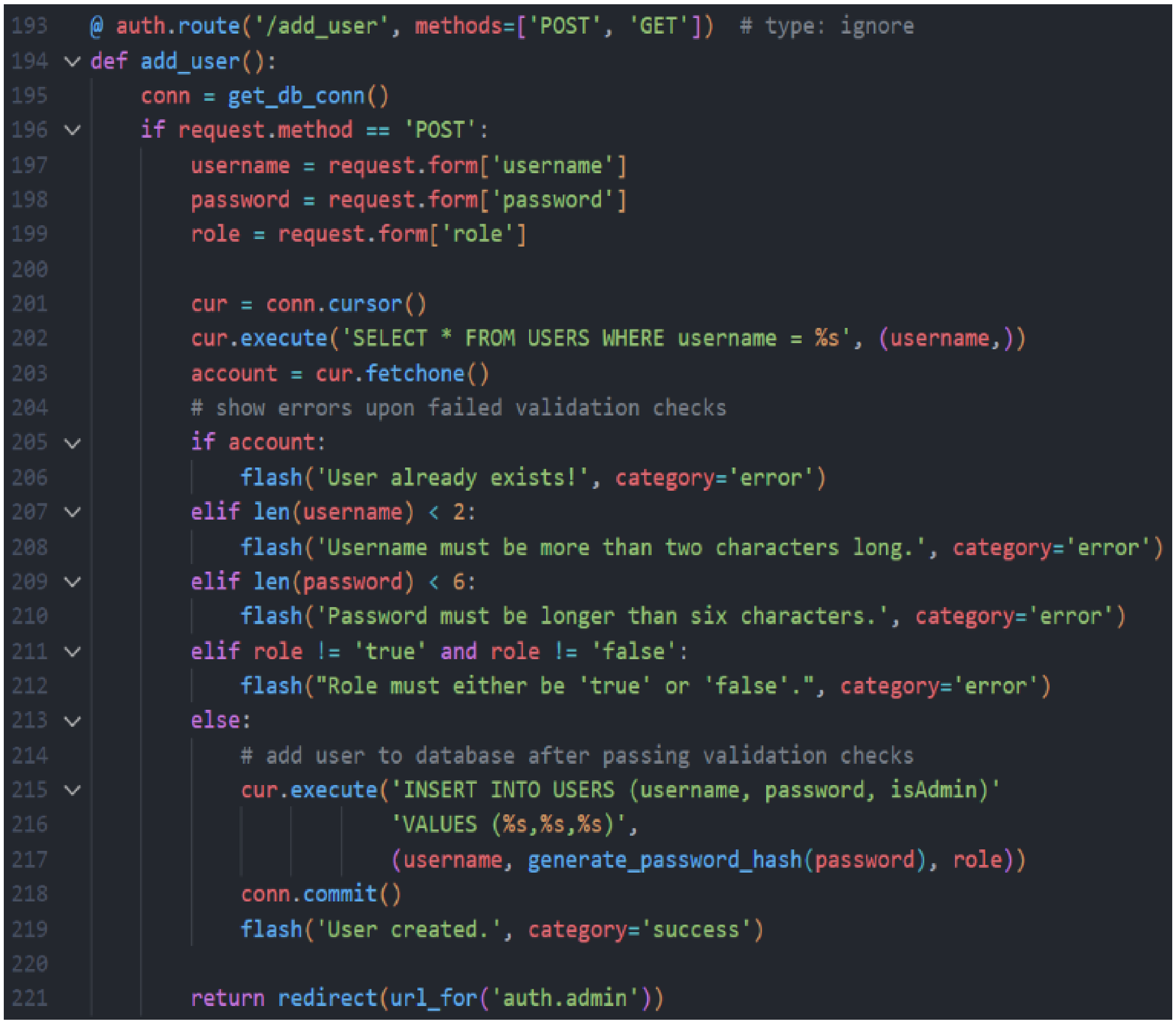
‘/logout’ redirects the user to the login page upon activation. The logout() function clears the users session and notifies the user that they have been logged out successfully.



```python
# Create Admin Page
@auth.route('/admin')
def admin():
    conn = get_db_conn()
    cur = conn.cursor()
    cur.execute('SELECT user_id, username, isAdmin FROM USERS;')
    users = cur.fetchall()
    return render_template("admin.html", user=users)
```

ADMINISTRATIVE

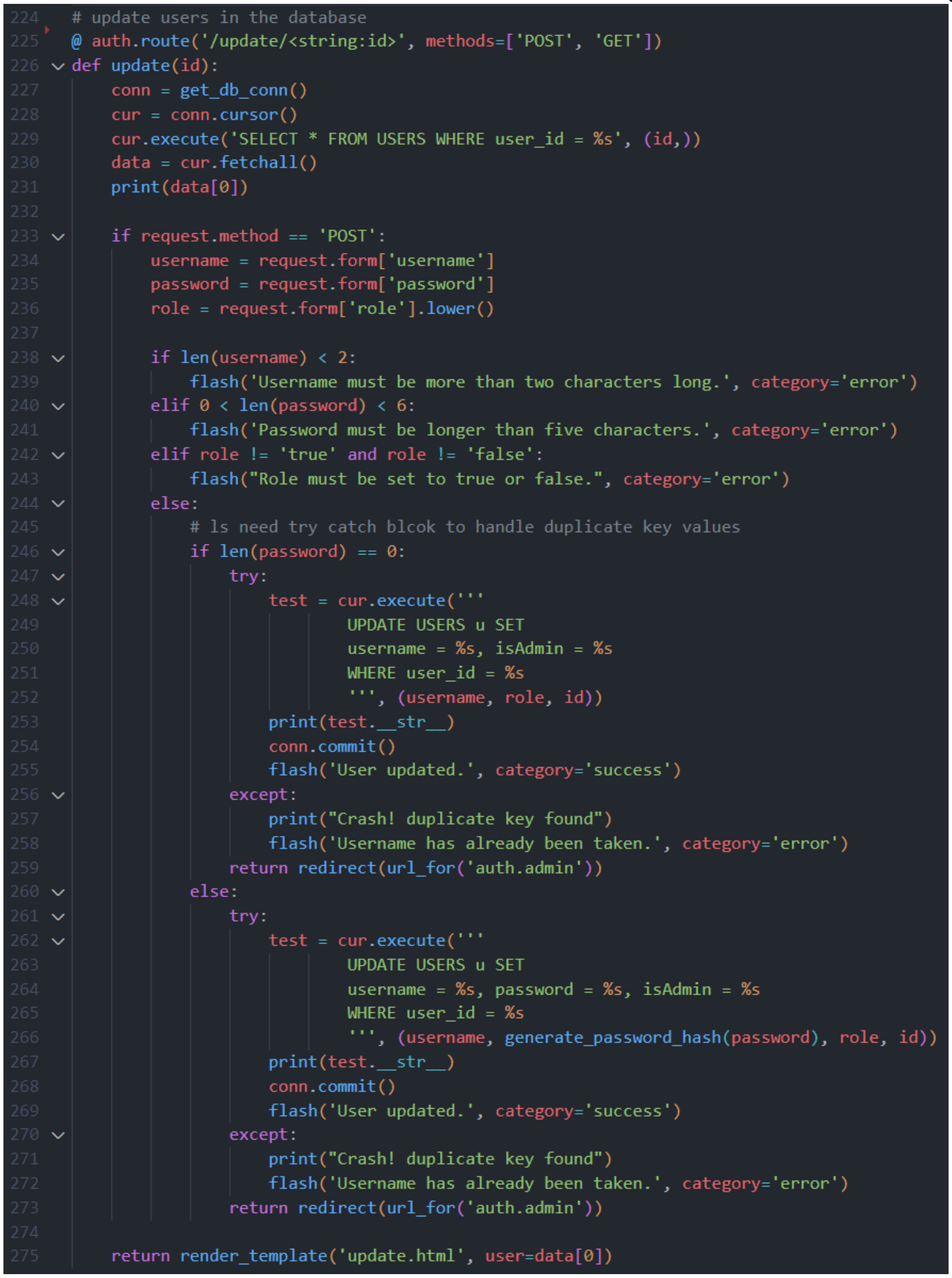
The '/admin' route serves as the destination for the admin panel. The admin() function executes an SQL query that selects the necessary data to fill and manipulate the USERS table on the admin.html webpage.



```python
@ auth.route('/add_user', methods=['POST', 'GET'])  # type: ignore
def add_user():
    conn = get_db_conn()
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        role = request.form['role']

        cur = conn.cursor()
        cur.execute('SELECT * FROM USERS WHERE username = %s', (username,))
        account = cur.fetchone()
        # show errors upon failed validation checks
        if account:
            flash('User already exists!', category='error')
        elif len(username) < 2:
            flash('Username must be more than two characters long.', category='error')
        elif len(password) < 6:
            flash('Password must be longer than six characters.', category='error')
        elif role != 'true' and role != 'false':
            flash("Role must either be 'true' or 'false'.", category='error')
        else:
            # add user to database after passing validation checks
            cur.execute('INSERT INTO USERS (username, password, isAdmin)'
                        'VALUES (%s,%s,%s)',
                        (username, generate_password_hash(password), role))
            conn.commit()
            flash('User created.', category='success')

        return redirect(url_for('auth.admin'))
```

'/add_user' is triggered by a form located on the admin panel. The form takes a username, password, and a role as input. The user creation form uses the same validation checks as the registration form for consistency across accounts. An admin should not be able to make accounts with "custom" credentials.

Upon form submission, the user is added to the database as long the user does not already exist, and all the constraints are met. The table located on the admin panel updates and displays the new user upon user creation.



```python
# update users in the database
@ auth.route('/update/<string:id>', methods=['POST', 'GET'])
def update(id):
    conn = get_db_conn()
    cur = conn.cursor()
    cur.execute('SELECT * FROM USERS WHERE user_id = %s', (id,))
    data = cur.fetchall()
    print(data[0])

    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        role = request.form['role'].lower()

        if len(username) < 2:
            flash('Username must be more than two characters long.', category='error')
        elif 0 < len(password) < 6:
            flash('Password must be longer than five characters.', category='error')
        elif role != 'true' and role != 'false':
            flash("Role must be set to true or false.", category='error')
        else:
            # ls need try catch blcok to handle duplicate key values
            if len(password) == 0:
                try:
                    test = cur.execute('''
                            UPDATE USERS u SET
                            username = %s, isAdmin = %s
                            WHERE user_id = %s
                            ''', (username, role, id))
                    print(test.__str__)
                    conn.commit()
                    flash('User updated.', category='success')
                except:
                    print("Crash! duplicate key found")
                    flash('Username has already been taken.', category='error')
                return redirect(url_for('auth.admin'))
            else:
                try:
                    test = cur.execute('''
                            UPDATE USERS u SET
                            username = %s, password = %s, isAdmin = %s
                            WHERE user_id = %s
                            ''', (username, generate_password_hash(password), role, id))
                    print(test.__str__)
                    conn.commit()
                    flash('User updated.', category='success')
                except:
                    print("Crash! duplicate key found")
                    flash('Username has already been taken.', category='error')
                return redirect(url_for('auth.admin'))

    return render_template('update.html', user=data[0])
```

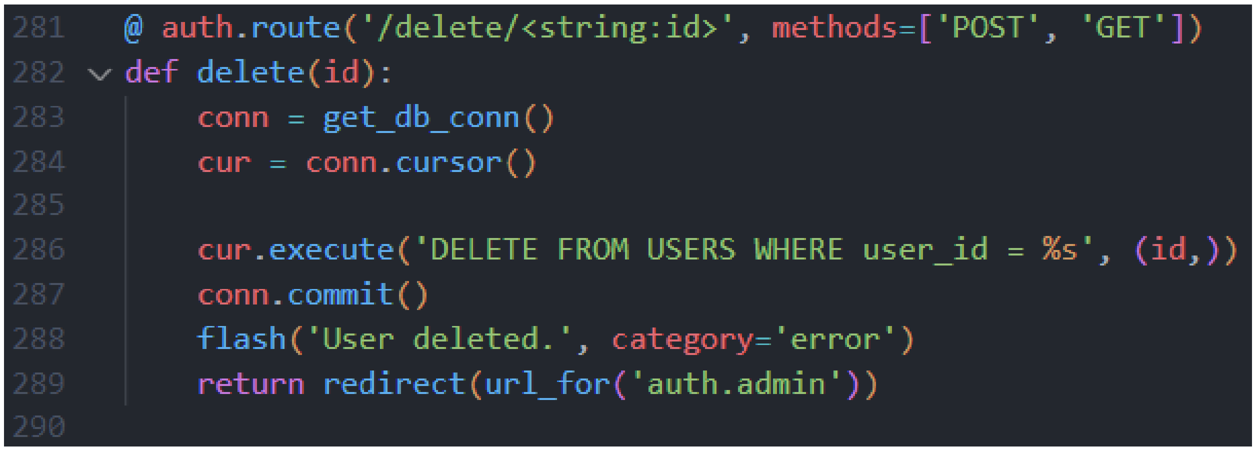
The ‘/update/<string:id>’ route is activated when an administrator begins editting a user in the USERS table through table interface on the admin panel.

The update(id) function updates the credentials of a user with the specified user_id. The user_id is specified based on which table row an administrator chooses to edit.

Like the registration and add_user forms, the update form also takes the same constraints into account.

If the request method is POST, and the provided credentials do not break any constraints, the user_id within the USERS table has its associated attributes updated to the newly provided username, password, and role.

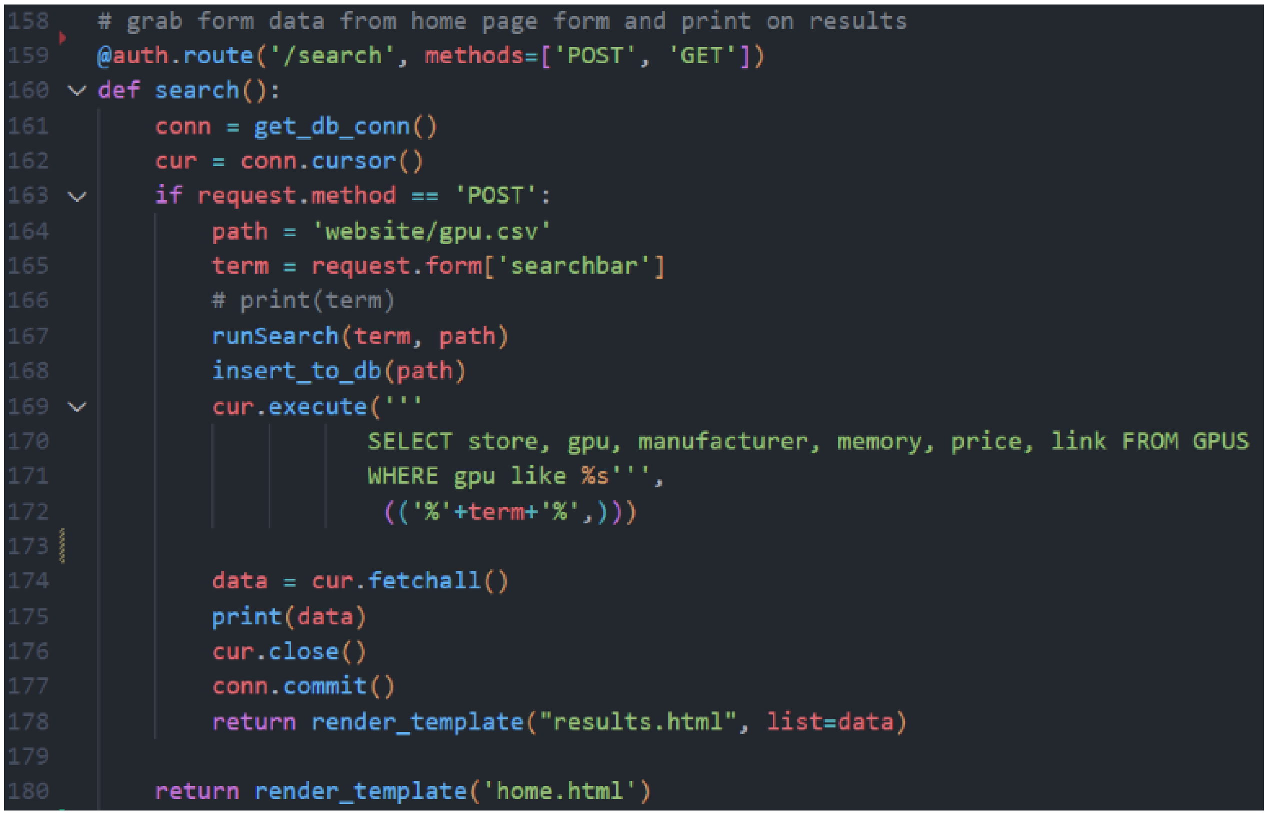
The table located on the admin panel is updated and displays the updated username and role associated with the user_id.



```python
@ auth.route('/delete/<string:id>', methods=['POST', 'GET'])
def delete(id):
    conn = get_db_conn()
    cur = conn.cursor()

    cur.execute('DELETE FROM USERS WHERE user_id = %s', (id,))
    conn.commit()
    flash('User deleted.', category='error')
    return redirect(url_for('auth.admin'))
```

The ‘/delete/<string:id>’ route is triggered when an administrator presses the “delete” button in the USERS table located on the admin panel. The associated user_id is deleted from the database and the table is updated to display these changes.

SEARCH



```
# grab form data from home page form and print on results
@auth.route('/search', methods=['POST', 'GET'])
def search():
    conn = get_db_conn()
    cur = conn.cursor()
    if request.method == 'POST':
        path = 'website/gpu.csv'
        term = request.form['searchbar']
        # print(term)
        runSearch(term, path)
        insert_to_db(path)
        cur.execute('''
                    SELECT store, gpu, manufacturer, memory, price, link FROM GPUS
                    WHERE gpu like %s''',
                     (('%'+term+'%',)))

        data = cur.fetchall()
        print(data)
        cur.close()
        conn.commit()
        return render_template("results.html", list=data)

    return render_template('home.html')
```

The '/search' route is triggered upon pressing the search button located on the home page.

The form associated to the search() function uses the input term to scrape a csv file of previously scraped information to display graphics cards that the user searched for.

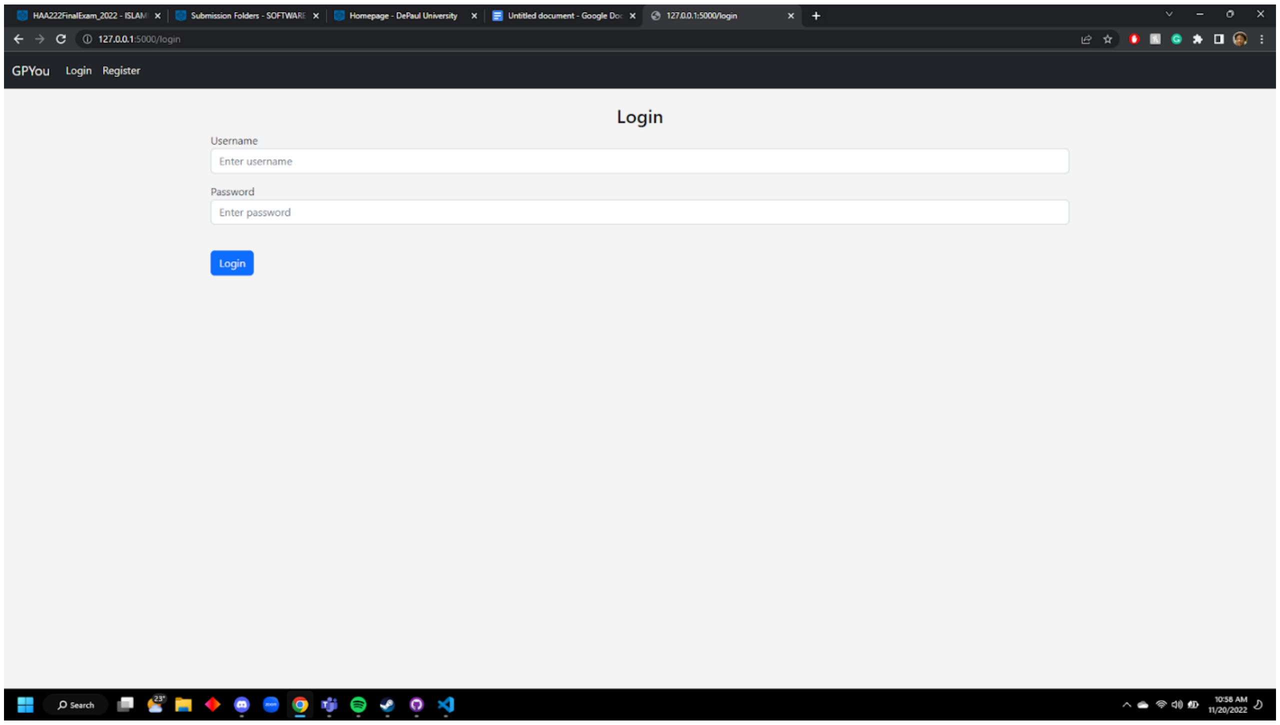
PRESENTATION LAYER

LOG-IN PAGE

The Log-In page features a simple design strategy allowing returning users to log in via the credentials they created when registering an account.

The functions are straightforward, only allowing the user to input a premade username and password combination. Once the "Login" Button is pressed they are redirected to a personalized home page

In this screen, the Nav Bar is persistent and allows the user to leave the login page and use the default home screen available to anyone without access to specialized features such as Favorites.

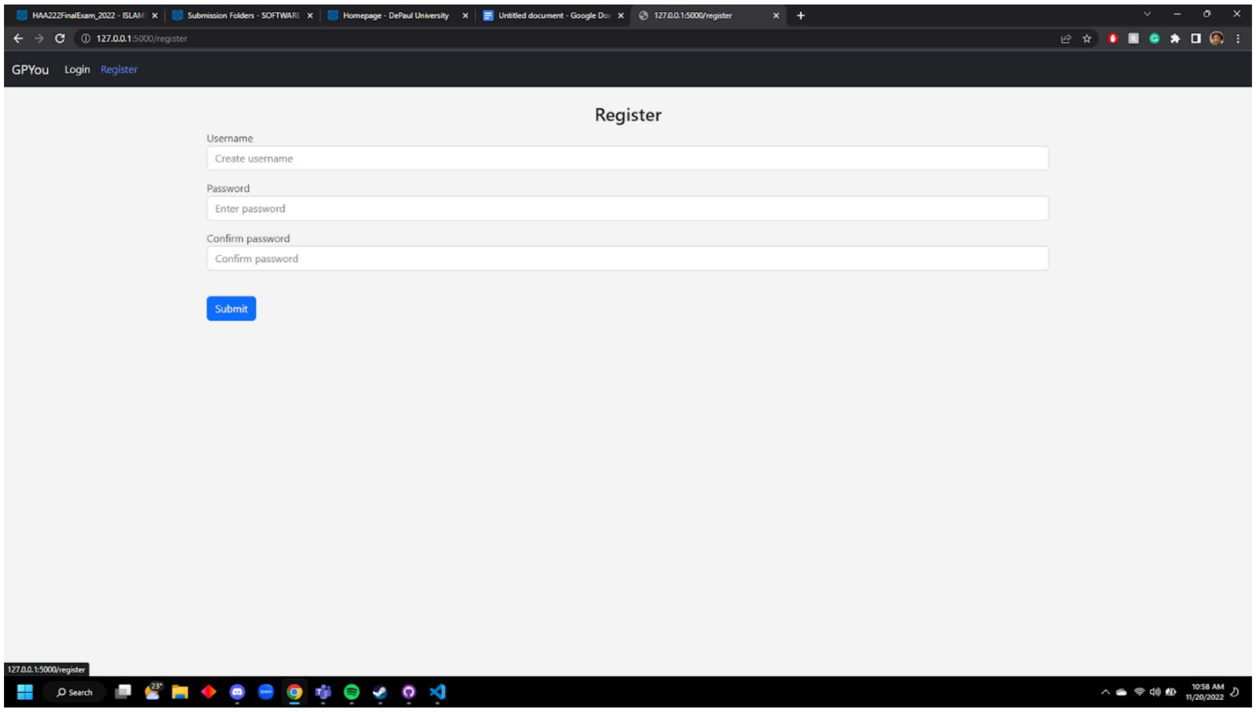


REGISTER PAGE

The Register Page features a simple design strategy allowing returning users to create a new account using unique identifiers.

The functions are straightforward, only allowing the user to input a unique username and password combination. Once the “Register” Button is pressed they are redirected to a personalized home page

In this screen, the Nav Bar is persistent and allows the user to leave the register page and use the default home screen available to anyone without access to specialized features such as Favorites.

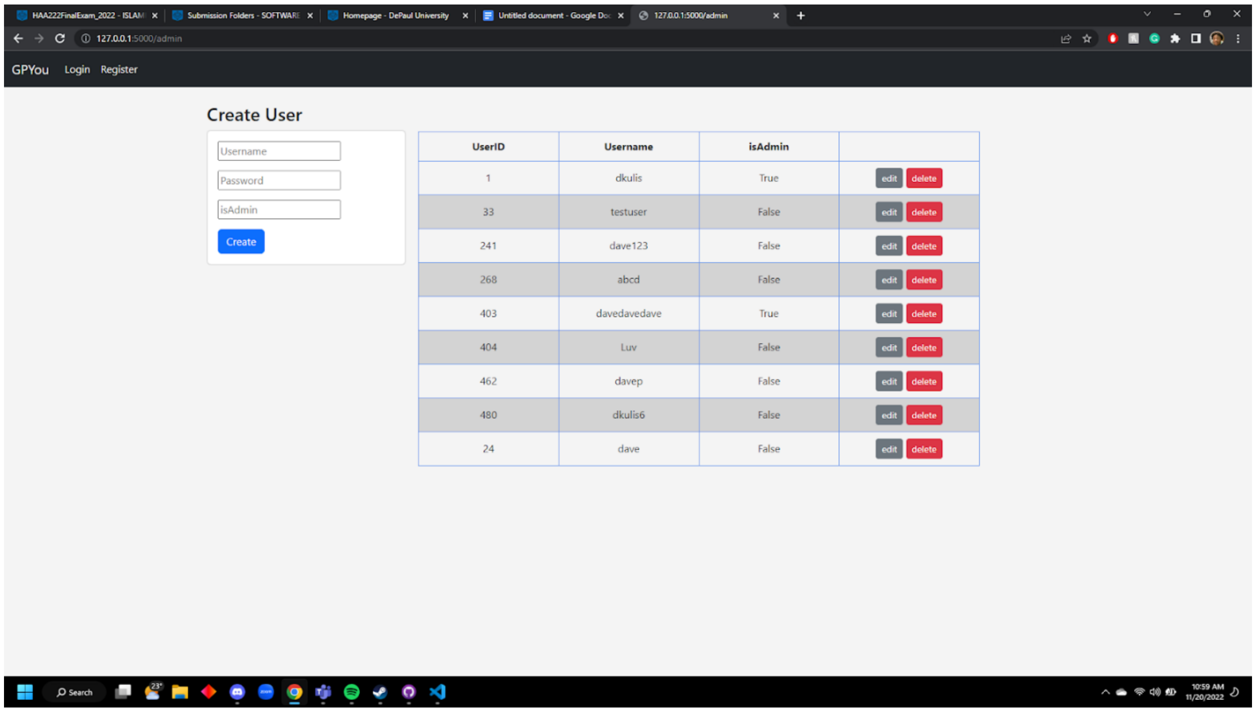


ADMIN PANEL

The Admin Panel Page features a simple design strategy allowing Admin Users to create a new account using unique identifiers as well as manage current accounts for users. In the event that credential needs to be updated, the admin user has the ability to change those values.

The functions are straightforward, only allowing the Admin User to input a unique username and password combination to create new accounts. One key feature about account creation is that a new account can be made admin immediately at creation. Once the “Create” Button is pressed a new account is created and is immediately ready for use.

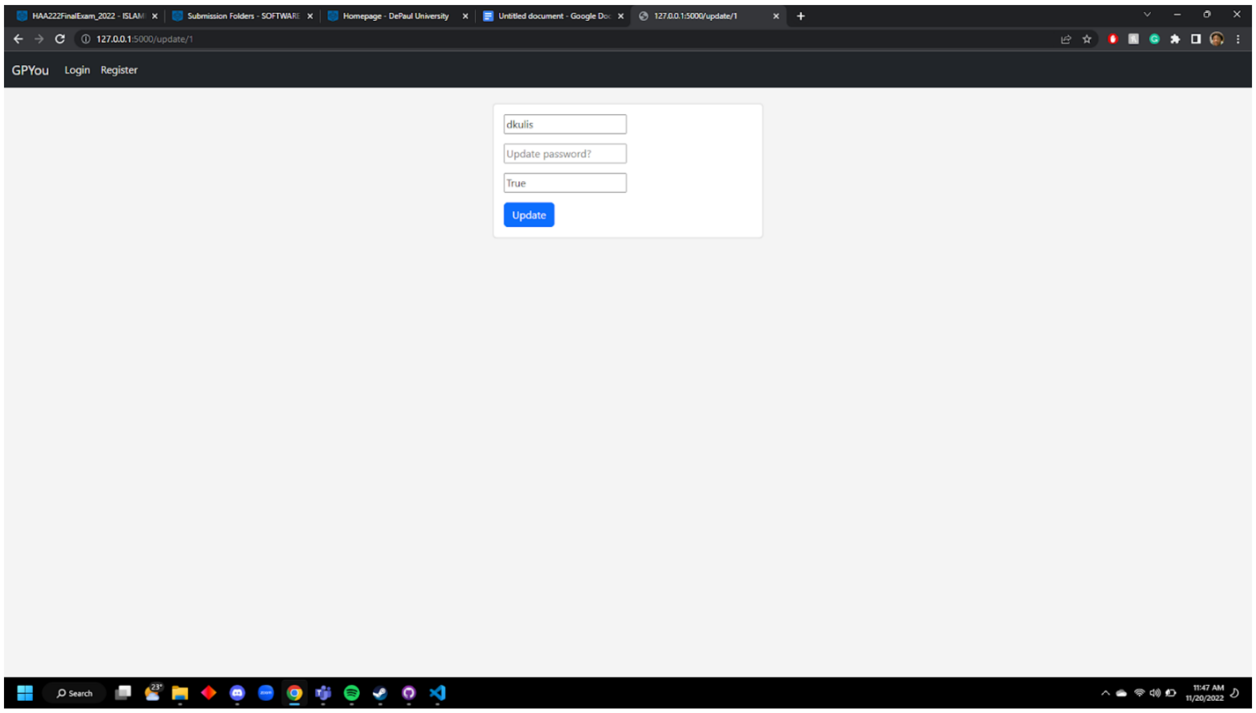
In the admin screen, the Nav Bar is persistent and allows the user to leave the admin page and use the default home screen available to anyone without access to specialized features such as Favorites. But if they please they can continue using the admin features and their personalized home screen


HAA222FinalExam_2022 - ISLAM
Submission Folders - SOFTWARE
Homepage - DePaul University
Untitled document - Google Doc
127.0.0.1:5000/admin
127.0.0.1:5000/admin
GPYou
Login
Register
Username
Password
isAdmin
Create

Create User

UserID	Username	isAdmin	
1	dkulis	True	edit delete
33	testuser	False	edit delete
241	dave123	False	edit delete
268	abcd	False	edit delete
403	davedavedave	True	edit delete
404	Luv	False	edit delete
462	davep	False	edit delete
480	dkulis6	False	edit delete
24	dave	False	edit delete

ADMIN PANEL CONT.



dkulis

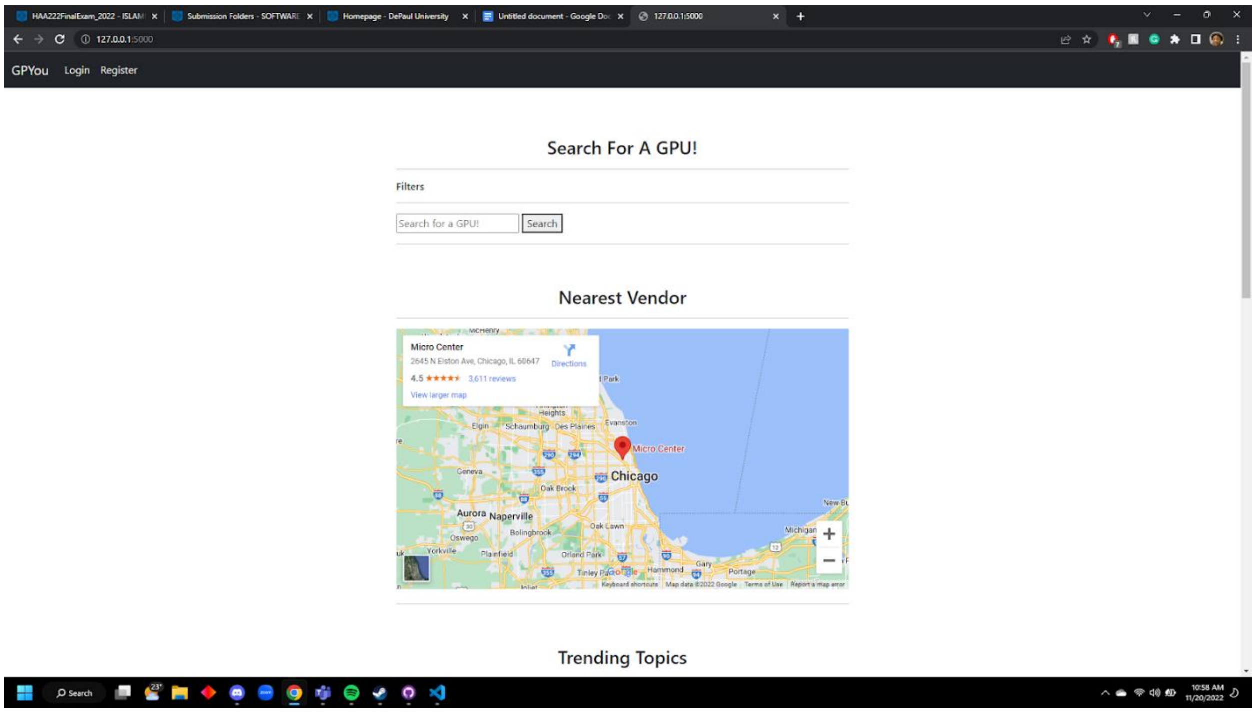
Update password?

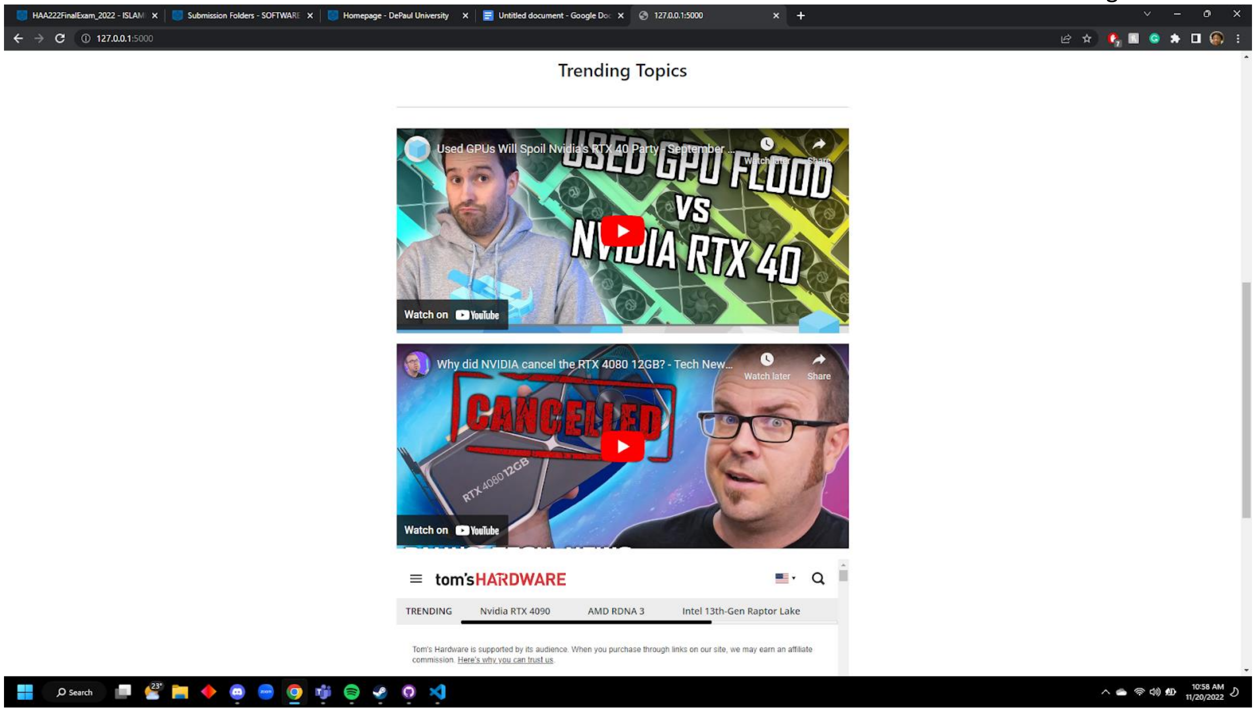
True

HOME SCREEN

The Home Screen contains three key elements; The search function, an embedded location of nearby hardware stores that sell GPUs, and a trending topics section that has embedded elements that manually get changed by admins.

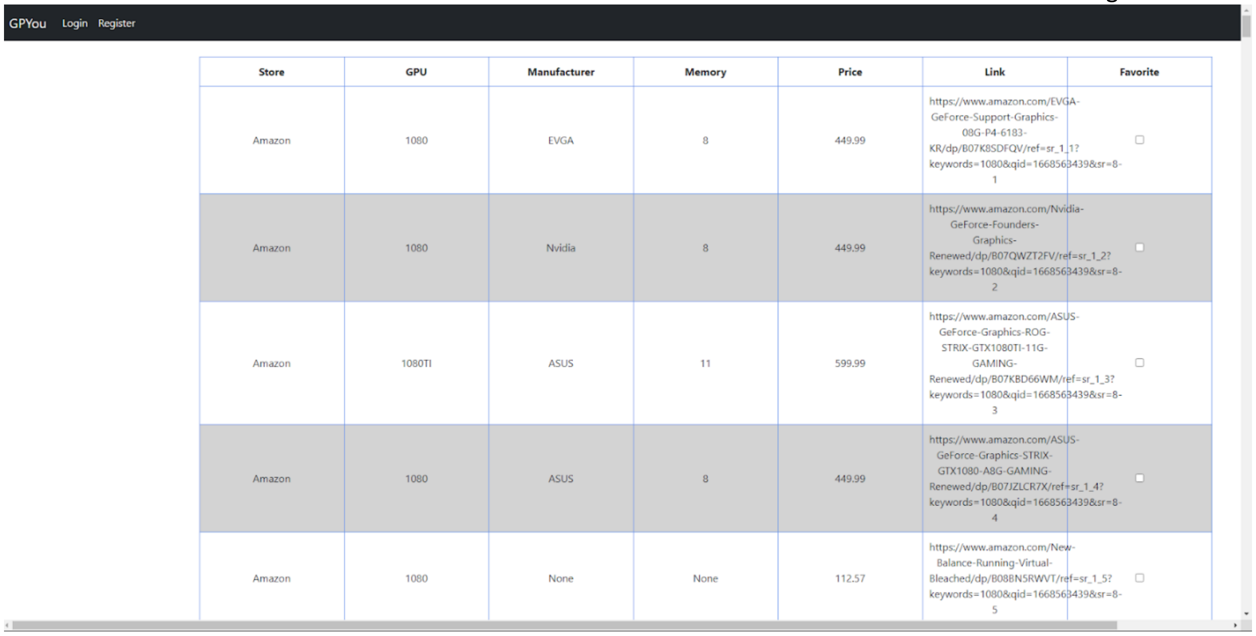
- Search Section
- Search for GPUs that have been added to a database via a web scraper protocol. Only works when you input a numerical value or combination of numbers + TI
- Once the "Search" button is pressed it redirects the user to a results page that contains the results of their query
- Nearest Vendor Section
- Displays the nearest vendor based on provided location data.
- Trending topics
- Displays four embedded elements based on admin discretion




Trending Topics
Used GPUs Will Spoil Nvidia's RTX 40 Party - September ...
USED GPU FLOOD VS NVIDIA RTX 40
Watch on YouTube
Why did NVIDIA cancel the RTX 4080 12GB? - Tech New...
CANCELLED
Watch later
Share
tom'sHARDWARE
TRENDING
Nvidia RTX 4090
AMD RDNA 3
Intel 13th-Gen Raptor Lake

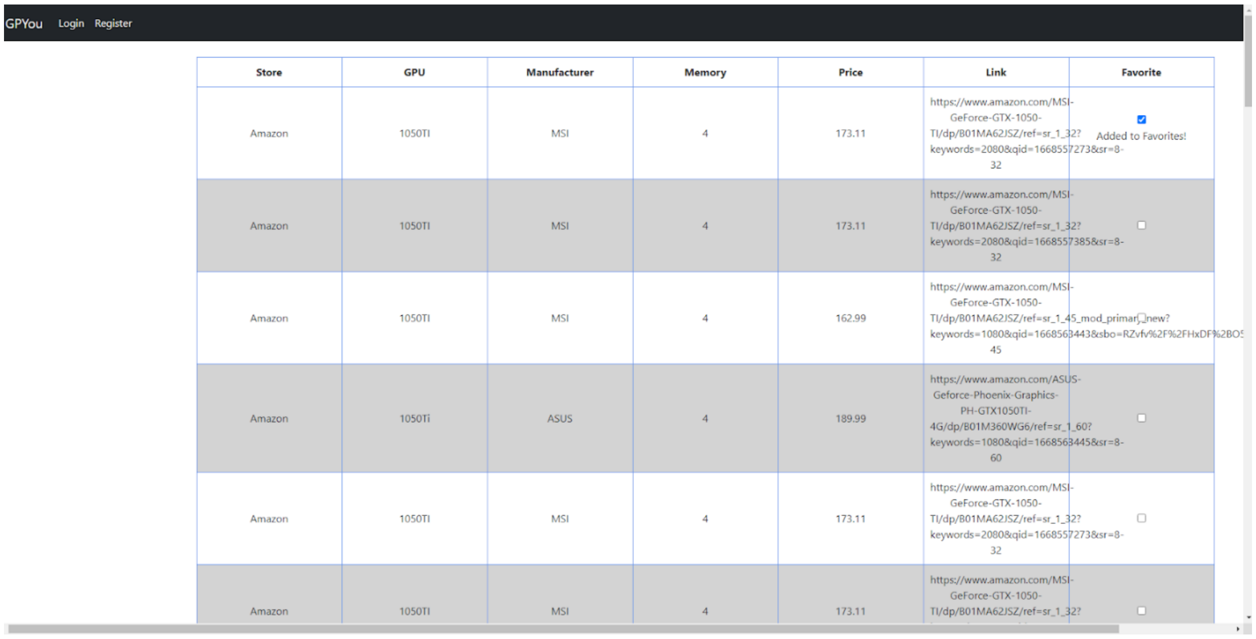
SEARCH RESULTS SCREEN

- The search results page yields a chart filled with results from the search query.
- There are several fields; Store, GPU, Manufacturer, Memory, Price, Link, and Favorite.
- Store Column tells you the Provider, Amazon or Newegg, of the GPU in question
- GPU Column tells you the name of the GPU, for example, if you searched for “1080TI” the only result in this column should be “1080TI”
- The manufacturer column should tell you the brand name of the GPU in question, common results in this column are EVGA, ASUS, and MSI
- The Memory section will tell you about the Memory of the GPU for example 4GB
- The Price section will tell you the price of the GPU in USD
- The link section will provide a direct link to the Amazon or Newegg page where you can purchase the GPU
- The favorite Column allows you to mark GPUs that pique the user's interest.



GPYou Login Register

Store	GPU	Manufacturer	Memory	Price	Link	Favorite
Amazon	1080	EVGA	8	449.99	https://www.amazon.com/EVGA-GeForce-Support-Graphics-08G-P4-6183-KR/dp/B07K8SDFQV/ref=sr_1_1?keywords=1080&qid=1668563439&sr=8-1	☐
Amazon	1080	Nvidia	8	449.99	https://www.amazon.com/Nvidia-GeForce-Founders-Graphics-Renewed/dp/B07QWZT2FV/ref=sr_1_2?keywords=1080&qid=1668563439&sr=8-2	☐
Amazon	1080TI	ASUS	11	599.99	https://www.amazon.com/ASUS-GeForce-Graphics-ROG-STRIX-GTX1080TI-11G-GAMING-Renewed/dp/B07KBD66WM/ref=sr_1_3?keywords=1080&qid=1668563439&sr=8-3	☐
Amazon	1080	ASUS	8	449.99	https://www.amazon.com/ASUS-GeForce-Graphics-STRIX-GTX1080-A8G-GAMING-Renewed/dp/B07JZLCR7X/ref=sr_1_4?keywords=1080&qid=1668563439&sr=8-4	☐
Amazon	1080	None	None	112.57	https://www.amazon.com/New-Balance-Running-Virtual-Bleached/dp/B08BN5RWVT/ref=sr_1_5?keywords=1080&qid=1668563439&sr=8-5	☐



GPYou Login Register

Store	GPU	Manufacturer	Memory	Price	Link	Favorite
Amazon	1050TI	MSI	4	173.11	https://www.amazon.com/MSI-GeForce-GTX-1050-TI/dp/B01MA62JSZ/ref=sr_1_32?keywords=2080&qid=1668557273&sr=8-32	☑ Added to Favorites!
Amazon	1050TI	MSI	4	173.11	https://www.amazon.com/MSI-GeForce-GTX-1050-TI/dp/B01MA62JSZ/ref=sr_1_32?keywords=2080&qid=1668557385&sr=8-32	☐
Amazon	1050TI	MSI	4	162.99	https://www.amazon.com/MSI-GeForce-GTX-1050-TI/dp/B01MA62JSZ/ref=sr_1_45_mod_primary_new?keywords=1080&qid=1668563443&sbo=RZvfv%2F%2FHxDF%2BO5 45	☐
Amazon	1050Ti	ASUS	4	189.99	https://www.amazon.com/ASUS-Geforce-Phoenix-Graphics-PH-GTX1050TI-4G/dp/B01M360WG6/ref=sr_1_60?keywords=1080&qid=1668563445&sr=8-60	☐
Amazon	1050TI	MSI	4	173.11	https://www.amazon.com/MSI-GeForce-GTX-1050-TI/dp/B01MA62JSZ/ref=sr_1_32?keywords=2080&qid=1668557273&sr=8-32	☐
Amazon	1050TI	MSI	4	173.11	https://www.amazon.com/MSI-GeForce-GTX-1050-TI/dp/B01MA62JSZ/ref=sr_1_32?	☐

CONTRIBUTIONS

LUV SHAH

- Project Manager
- Oversaw scheduling and d
- eadlines for team objectives
- Delegated roles and responsibilities among team members
- Organized and hosted meetings twice weekly
- Reported on performance with team members
- Communicated with teammates on critical topics
- Assisted with miscellaneous bugs / fixes

KOSTA KYRIAKOULIS

- Design Manager
- Built Newegg Scraper functionality
- Built Backend Database Connection layer
- Demo 1,2,3 Designs
- Database Design Schema
- Architecture Design
- Assisted with Flask and Django implementation.

DAVE PATEL

- Testing Manager
- Hosting Website onto EC2
- Refactor code
- Amazon Scrapper
- Paser and CSV writer
- Login Sessions
- Test Suit
- Assisted in creating the initial code files for frontend

DAVID KUSLIS

- Requirements Manager
- Created initial skeleton code for frontend
- Designed and created admin panel
- Programmed functionality of admin panels

- Create log-in sessions
- Assisted in designing back-end scraper
- Created Log In, and Registration panel
- Added functionality to log in and registration panel

JAYDEN GODBOLD

- Designed and create majority of the front end
- Created Home page
- Created and design the result page
- Created Navbar with tabs
- Refactored existing designs into new ones
- Implemented Front-end to back-end communication