**PROJECT NAME: “ARTBOX which is designed specifically for TATTOO ARTIST AND COLLECTORS.”**

**Step 1: Set Up Flask Application**

**First, set up a virtual environment and install Flask.**

python -m venv venv

source venv/bin/activate # On Windows use `venv\Scripts\activate`

pip install Flask

**Step 2: Create the Flask Application**

**Create a file called app.py:**

from flask import Flask, render\_template, request, jsonify

app = Flask(\_\_name\_\_)

@app.route('/')

def home():

return render\_template('index.html')

@app.route('/generate', methods=['POST'])

def generate():

prompt = request.form['prompt']

# Placeholder for AI generation logic

# In a real scenario, you would integrate with an AI model here.

generated\_image\_url = f"static/images/generated\_image.png"

return jsonify({'image\_url': generated\_image\_url})

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

**Step 3: Create the Front-End**

**Create a templates folder and an index.html file inside it:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Artbox - Tattoo Designs</title>

<link rel="stylesheet" href="{{ url\_for('static', filename='styles.css') }}">

</head>

<body>

<div class="container">

<h1>Artbox</h1>

<form id="generate-form">

<label for="prompt">Enter Tattoo Description:</label>

<input type="text" id="prompt" name="prompt" required>

<button type="submit">Generate</button>

</form>

<div id="result">

<img id="generated-image" src="" alt="Generated Tattoo">

</div>

</div>

<script>

document.getElementById('generate-form').addEventListener('submit', function(e) {

e.preventDefault();

const prompt = document.getElementById('prompt').value;

fetch('/generate', {

method: 'POST',

headers: {

'Content-Type': 'application/json'

},

body: JSON.stringify({ prompt: prompt })

})

.then(response => response.json())

.then(data => {

document.getElementById('generated-image').src = data.image\_url;

});

});

</script>

</body>

</html>

**Step 4: Add CSS for Styling**

**Create a static folder and a styles.css file inside it:**

body {

font-family: Arial, sans-serif;

background-color: #f9f9f9;

display: flex;

justify-content: center;

align-items: center;

height: 100vh;

margin: 0;

}

.container {

background-color: #fff;

padding: 20px;

border-radius: 8px;

box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

text-align: center;

}

h1 {

margin-bottom: 20px;

color: #333;

}

form {

margin-bottom: 20px;

}

label {

display: block;

margin-bottom: 8px;

color: #666;

}

input[type="text"] {

width: 100%;

padding: 8px;

margin-bottom: 12px;

border: 1px solid #ccc;

border-radius: 4px;

}

button {

padding: 10px 20px;

border: none;

border-radius: 4px;

background-color: #28a745;

color: white;

cursor: pointer;

}

button:hover {

background-color: #218838;

}

#result {

margin-top: 20px;

}

#generated-image {

max-width: 100%;

height: auto;

border-radius: 4px;

}

**Step 5: Run the Flask Application**

**Run the Flask application by executing the following command in your terminal:**

python app.py

**Step 6: Integrate AI Model for Generating Tattoo Designs**

**Update the app.py file to include a function that simulates the AI generation:**

from flask import Flask, render\_template, request, jsonify

import os

import random

app = Flask(\_\_name\_\_)

# Placeholder function to simulate AI image generation

def generate\_ai\_image(prompt):

# Simulate image generation by randomly selecting an image from a predefined list

images = [

"static/images/tattoo1.png",

"static/images/tattoo2.png",

"static/images/tattoo3.png"

]

return random.choice(images)

@app.route('/')

def home():

return render\_template('index.html')

@app.route('/generate', methods=['POST'])

def generate():

data = request.get\_json()

prompt = data['prompt']

generated\_image\_url = generate\_ai\_image(prompt)

return jsonify({'image\_url': generated\_image\_url})

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

**Step 7: Update Front-End JavaScript to Handle JSON Data**

**Update the index.html file's JavaScript code to send and receive JSON data:**

<script>

document.getElementById('generate-form').addEventListener('submit', function(e) {

e.preventDefault();

const prompt = document.getElementById('prompt').value;

fetch('/generate', {

method: 'POST',

headers: {

'Content-Type': 'application/json'

},

body: JSON.stringify({ prompt: prompt })

})

.then(response => response.json())

.then(data => {

document.getElementById('generated-image').src = data.image\_url;

});

});

</script>

**Step 8: Membership Levels**

**membership.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Artbox - Membership</title>

<link rel="stylesheet" href="{{ url\_for('static', filename='styles.css') }}">

</head>

<body>

<div class="container">

<h1>Membership Plans</h1>

<div class="membership-plan">

<h2>Basic</h2>

<p>$9.99/month</p>

<p>Few Images, Very Less Storage</p>

<button>Select</button>

</div>

<div class="membership-plan">

<h2>Premium</h2>

<p>$39.99/month</p>

<p>Good Images, Good Storage</p>

<button>Select</button>

</div>

<div class="membership-plan">

<h2>Pro</h2>

<p>$69.99/month</p>

<p>More Images, More Storage</p>

<button>Select</button>

</div>

</div>

</body>

</html>

**Step 9: Add Route for Membership Page**

**Update app.py to include the route for the membership page:**

@app.route('/membership')

def membership():

return render\_template('membership.html')

**Step 10: Navigation and Layout**

**Update index.html to include navigation to the membership page:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Artbox - Tattoo Designs</title>

<link rel="stylesheet" href="{{ url\_for('static', filename='styles.css') }}">

</head>

<body>

<div class="container">

<h1>Artbox</h1>

<nav>

<a href="/">Home</a> |

<a href="/membership">Membership</a>

</nav>

<form id="generate-form">

<label for="prompt">Enter Tattoo Description:</label>

<input type="text" id="prompt" name="prompt" required>

<button type="submit">Generate</button>

</form>

<div id="result">

<img id="generated-image" src="" alt="Generated Tattoo">

</div>

</div>

<script>

document.getElementById('generate-form').addEventListener('submit', function(e) {

e.preventDefault();

const prompt = document.getElementById('prompt').value;

fetch('/generate', {

method: 'POST',

headers: {

'Content-Type': 'application/json'

},

body: JSON.stringify({ prompt: prompt })

})

.then(response => response.json())

.then(data => {

document.getElementById('generated-image').src = data.image\_url;

});

});

</script>

</body>

</html>

**Step 11: Finalize CSS for Membership Plans**

**Update styles.css to style the membership plans:**

.membership-plan {

border: 1px solid #ccc;

padding: 20px;

margin: 10px;

border-radius: 8px;

}

.membership-plan h2 {

margin: 0 0 10px;

}

.membership-plan p {

margin: 5px 0;

}

.membership-plan button {

padding: 10px 20px;

border: none;

border-radius: 4px;

background-color: #007bff;

color: white;

cursor: pointer;

}

.membership-plan button:hover {

background-color: #0056b3;

}

**Step 12: Implement User Authentication**

**To handle user authentication, you can use Flask-Login. First, install Flask-Login:**

pip install flask-login

Create a user.py file to define the User model:

from flask\_login import UserMixin

class User(UserMixin):

def \_\_init\_\_(self, id, username, password):

self.id = id

self.username = username

self.password = password

# Simulated database

users = {

1: User(1, "user1", "password1"),

2: User(2, "user2", "password2")

}

def get\_user(user\_id):

return users.get(int(user\_id))

def find\_user\_by\_username(username):

for user in users.values():

if user.username == username:

return user

return None

**Step 13: Update app.py for Authentication**

**Update app.py to include routes for login and logout, and protect certain routes with login required:**

from flask import Flask, render\_template, request, jsonify, redirect, url\_for

from flask\_login import LoginManager, login\_user, login\_required, logout\_user, current\_user

from user import User, get\_user, find\_user\_by\_username

app = Flask(\_\_name\_\_)

app.secret\_key = 'supersecretkey'

login\_manager = LoginManager()

login\_manager.init\_app(app)

login\_manager.login\_view = 'login'

@login\_manager.user\_loader

def load\_user(user\_id):

return get\_user(user\_id)

@app.route('/')

def home():

return render\_template('index.html')

@app.route('/login', methods=['GET', 'POST'])

def login():

if request.method == 'POST':

username = request.form['username']

password = request.form['password']

user = find\_user\_by\_username(username)

if user and user.password == password:

login\_user(user)

return redirect(url\_for('home'))

return render\_template('login.html')

@app.route('/logout')

@login\_required

def logout():

logout\_user()

return redirect(url\_for('login'))

@app.route('/generate', methods=['POST'])

@login\_required # Ensure the user is logged in to access this route

def generate():

data = request.get\_json()

prompt = data['prompt']

generated\_image\_url = generate\_ai\_image(prompt)

return jsonify({'image\_url': generated\_image\_url})

@app.route('/membership')

@login\_required

def membership():

return render\_template('membership.html')

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

**Step 14: Create Login and Logout Pages**

**Create login.html in the templates folder:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Login - Artbox</title>

<link rel="stylesheet" href="{{ url\_for('static', filename='styles.css') }}">

</head>

<body>

<div class="container">

<h1>Login</h1>

<form method="POST" action="{{ url\_for('login') }}">

<label for="username">Username:</label>

<input type="text" id="username" name="username" required>

<label for="password">Password:</label>

<input type="password" id="password" name="password" required>

<button type="submit">Login</button>

</form>

</div>

</body>

</html>

**Update index.html to show login/logout links based on the user's authentication status:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Artbox - Tattoo Designs</title>

<link rel="stylesheet" href="{{ url\_for('static', filename='styles.css') }}">

</head>

<body>

<div class="container">

<h1>Artbox</h1>

<nav>

<a href="/">Home</a> |

{% if current\_user.is\_authenticated %}

<a href="/membership">Membership</a> |

<a href="/logout">Logout</a>

{% else %}

<a href="/login">Login</a>

{% endif %}

</nav>

<form id="generate-form">

<label for="prompt">Enter Tattoo Description:</label>

<input type="text" id="prompt" name="prompt" required>

<button type="submit">Generate</button>

</form>

<div id="result">

<img id="generated-image" src="" alt="Generated Tattoo">

</div>

</div>

<script>

document.getElementById('generate-form').addEventListener('submit', function(e) {

e.preventDefault();

const prompt = document.getElementById('prompt').value;

fetch('/generate', {

method: 'POST',

headers: {

'Content-Type': 'application/json'

},

body: JSON.stringify({ prompt: prompt })

})

.then(response => response.json())

.then(data => {

document.getElementById('generated-image').src = data.image\_url;

});

});

</script>

</body>

</html>

**Step 15: Implement Membership Management**

**Update user.py to include membership levels:**

class User(UserMixin):

def \_\_init\_\_(self, id, username, password, membership):

self.id = id

self.username = username

self.password = password

self.membership = membership

# Simulated database

users = {

1: User(1, "user1", "password1", "Basic"),

2: User(2, "user2", "password2", "Premium")

}

# Memberships: Basic, Premium, Pro

def get\_user(user\_id):

return users.get(int(user\_id))

def find\_user\_by\_username(username):

for user in users.values():

if user.username == username:

return user

return None

**Step 16: Implement Membership Logic in the Application**

**Update the app.py file to track the number of images generated by each user:**

from flask import Flask, render\_template, request, jsonify, redirect, url\_for, session

from flask\_login import LoginManager, login\_user, login\_required, logout\_user, current\_user

from user import User, get\_user, find\_user\_by\_username

from datetime import datetime, timedelta

app = Flask(\_\_name\_\_)

app.secret\_key = 'supersecretkey'

login\_manager = LoginManager()

login\_manager.init\_app(app)

login\_manager.login\_view = 'login'

# Simulated in-memory store for user image generation counts

user\_image\_counts = {1: 0, 2: 0}

# Limits based on membership levels

membership\_limits = {

"Basic": 10,

"Premium": 50,

"Pro": 100

}

@login\_manager.user\_loader

def load\_user(user\_id):

return get\_user(user\_id)

@app.route('/')

def home():

return render\_template('index.html')

@app.route('/login', methods=['GET', 'POST'])

def login():

if request.method == 'POST':

username = request.form['username']

password = request.form['password']

user = find\_user\_by\_username(username)

if user and user.password == password:

login\_user(user)

return redirect(url\_for('home'))

return render\_template('login.html')

@app.route('/logout')

@login\_required

def logout():

logout\_user()

return redirect(url\_for('login'))

@app.route('/generate', methods=['POST'])

@login\_required

def generate():

data = request.get\_json()

prompt = data['prompt']

user\_id = current\_user.id

user\_membership = current\_user.membership

max\_images = membership\_limits[user\_membership]

if user\_image\_counts[user\_id] >= max\_images:

return jsonify({'error': 'Image generation limit reached for your membership level.'}), 403

generated\_image\_url = generate\_ai\_image(prompt)

user\_image\_counts[user\_id] += 1

return jsonify({'image\_url': generated\_image\_url})

@app.route('/membership')

@login\_required

def membership():

return render\_template('membership.html')

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

**Step 17: Update the Front-End to Handle Membership and Limits**

**Update index.html to handle error messages:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Artbox - Tattoo Designs</title>

<link rel="stylesheet" href="{{ url\_for('static', filename='styles.css') }}">

</head>

<body>

<div class="container">

<h1>Artbox</h1>

<nav>

<a href="/">Home</a> |

{% if current\_user.is\_authenticated %}

<a href="/membership">Membership</a> |

<a href="/logout">Logout</a>

{% else %}

<a href="/login">Login</a>

{% endif %}

</nav>

<form id="generate-form">

<label for="prompt">Enter Tattoo Description:</label>

<input type="text" id="prompt" name="prompt" required>

<button type="submit">Generate</button>

</form>

<div id="result">

<img id="generated-image" src="" alt="Generated Tattoo">

</div>

<div id="error-message" style="color: red;"></div>

</div>

<script>

document.getElementById('generate-form').addEventListener('submit', function(e) {

e.preventDefault();

const prompt = document.getElementById('prompt').value;

fetch('/generate', {

method: 'POST',

headers: {

'Content-Type': 'application/json'

},

body: JSON.stringify({ prompt: prompt })

})

.then(response => response.json().then(data => ({status: response.status, body: data})))

.then(response => {

if (response.status === 200) {

document.getElementById('generated-image').src = response.body.image\_url;

document.getElementById('error-message').textContent = '';

} else {

document.getElementById('error-message').textContent = response.body.error;

}

});

});

</script>

</body>

</html>

**Step 18: Create a User Profile Page**

**Create profile.html in the templates folder:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Profile - Artbox</title>

<link rel="stylesheet" href="{{ url\_for('static', filename='styles.css') }}">

</head>

<body>

<div class="container">

<h1>Profile</h1>

<p>Username: {{ current\_user.username }}</p>

<p>Membership: {{ current\_user.membership }}</p>

<p>Images Generated: {{ image\_count }}</p>

<p>Remaining Quota: {{ quota - image\_count }}</p>

<nav>

<a href="/">Home</a> |

<a href="/membership">Membership</a> |

<a href="/logout">Logout</a>

</nav>

</div>

</body>

</html>

**Step 19: Add Route for Profile Page**

**Update app.py to include the route for the profile page:**

@app.route('/profile')

@login\_required

def profile():

user\_id = current\_user.id

image\_count = user\_image\_counts[user\_id]

quota = membership\_limits[current\_user.membership]

return render\_template('profile.html', image\_count=image\_count, quota=quota)

Update index.html to include a link to the profile page:

<nav>

<a href="/">Home</a> |

<a href="/profile">Profile</a> |

{% if current\_user.is\_authenticated %}

<a href="/membership">Membership</a> |

<a href="/logout">Logout</a>

{% else %}

<a href="/login">Login</a>

{% endif %}

</nav>

**Step 20: Database Integration**

**First, install the required packages:**

pip install Flask-SQLAlchemy

**Step 21: Configure SQLAlchemy**

**Update app.py to include SQLAlchemy configuration and models for User and ImageGeneration:**

from flask import Flask, render\_template, request, jsonify, redirect, url\_for

from flask\_sqlalchemy import SQLAlchemy

from flask\_login import LoginManager, login\_user, login\_required, logout\_user, current\_user, UserMixin

from datetime import datetime

app = Flask(\_\_name\_\_)

app.secret\_key = 'supersecretkey'

app.config['SQLALCHEMY\_DATABASE\_URI'] = 'sqlite:///artbox.db'

db = SQLAlchemy(app)

login\_manager = LoginManager()

login\_manager.init\_app(app)

login\_manager.login\_view = 'login'

class User(UserMixin, db.Model):

id = db.Column(db.Integer, primary\_key=True)

username = db.Column(db.String(150), unique=True, nullable=False)

password = db.Column(db.String(150), nullable=False)

membership = db.Column(db.String(50), nullable=False)

class ImageGeneration(db.Model):

id = db.Column(db.Integer, primary\_key=True)

user\_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)

timestamp = db.Column(db.DateTime, default=datetime.utcnow)

@login\_manager.user\_loader

def load\_user(user\_id):

return User.query.get(int(user\_id))

@app.before\_first\_request

def create\_tables():

db.create\_all()

@app.route('/')

def home():

return render\_template('index.html')

@app.route('/login', methods=['GET', 'POST'])

def login():

if request.method == 'POST':

username = request.form['username']

password = request.form['password']

user = User.query.filter\_by(username=username).first()

if user and user.password == password:

login\_user(user)

return redirect(url\_for('home'))

return render\_template('login.html')

@app.route('/logout')

@login\_required

def logout():

logout\_user()

return redirect(url\_for('login'))

@app.route('/generate', methods=['POST'])

@login\_required

def generate():

data = request.get\_json()

prompt = data['prompt']

user\_id = current\_user.id

user\_membership = current\_user.membership

max\_images = membership\_limits[user\_membership]

image\_count = ImageGeneration.query.filter\_by(user\_id=user\_id).count()

if image\_count >= max\_images:

return jsonify({'error': 'Image generation limit reached for your membership level.'}), 403

generated\_image\_url = generate\_ai\_image(prompt)

new\_image = ImageGeneration(user\_id=user\_id)

db.session.add(new\_image)

db.session.commit()

return jsonify({'image\_url': generated\_image\_url})

@app.route('/profile')

@login\_required

def profile():

user\_id = current\_user.id

image\_count = ImageGeneration.query.filter\_by(user\_id=user\_id).count()

quota = membership\_limits[current\_user.membership]

return render\_template('profile.html', image\_count=image\_count, quota=quota)

@app.route('/membership')

@login\_required

def membership():

return render\_template('membership.html')

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

**Step 22: Update User Creation and Management**

**Add a route to handle user registration and update the login route to create initial users for testing:**

@app.route('/register', methods=['GET', 'POST'])

def register():

if request.method == 'POST':

username = request.form['username']

password = request.form['password']

membership = request.form['membership']

existing\_user = User.query.filter\_by(username=username).first()

if existing\_user:

return 'User already exists!'

new\_user = User(username=username, password=password, membership=membership)

db.session.add(new\_user)

db.session.commit()

return redirect(url\_for('login'))

return render\_template('register.html')

**Create register.html in the templates folder:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Register - Artbox</title>

<link rel="stylesheet" href="{{ url\_for('static', filename='styles.css') }}">

</head>

<body>

<div class="container">

<h1>Register</h1>

<form method="POST" action="{{ url\_for('register') }}">

<label for="username">Username:</label>

<input type="text" id="username" name="username" required>

<label for="password">Password:</label>

<input type="password" id="password" name="password" required>

<label for="membership">Membership:</label>

<select id="membership" name="membership" required>

<option value="Basic">Basic</option>

<option value="Premium">Premium</option>

<option value="Pro">Pro</option>

</select>

<button type="submit">Register</button>

</form>

</div>

</body>

</html>

**Step 23: Testing**

**After setting up everything, run the application:**

python app.py

**Visit the registration page at** [**http://127.0.0.1:5000/**](http://127.0.0.1:5000/) **register to create a new user. Then log in using that user and test the application by generating images and checking your profile page.**

**Step 24: Enhance AI Integration and Payment Gateway**

**For AI image generation, you need to replace the generate\_ai\_image function with actual AI model integration, such as calling an external API or using a pre-trained model locally.**

**Step 25: AI Image Generation Integration**

**First, sign up for the service and get an API key. Then, install the requests library if you haven't already:**

pip install requests

Update the generate\_ai\_image function in app.py to use the DeepAI API:

import requests

def generate\_ai\_image(prompt):

api\_key = 'YOUR\_DEEPAI\_API\_KEY'

response = requests.post(

"https://api.deepai.org/api/text2img",

data={'text': prompt},

headers={'api-key': api\_key}

)

response\_json = response.json()

return response\_json['output\_url']

**Step 26: Payment Gateway Integration with Stripe**

**Integrating a payment gateway like Stripe involves setting up subscription plans, handling payments, and managing user membership status. Here’s a step-by-step guide to integrating Stripe:**

**Sign up for Stripe and get your API keys.**

**Install the Stripe library:**

pip install stripe

**Configure Stripe in your application:**

import stripe

stripe.api\_key = 'YOUR\_STRIPE\_SECRET\_KEY'

Create subscription plans in your Stripe dashboard (Basic, Premium, Pro).

**Add Routes for Subscription Management:**

@app.route('/create-checkout-session', methods=['POST'])

@login\_required

def create\_checkout\_session():

try:

session = stripe.checkout.Session.create(

payment\_method\_types=['card'],

line\_items=[{

'price': request.form['price\_id'],

'quantity': 1,

}],

mode='subscription',

success\_url=url\_for('success', \_external=True) + '?session\_id={CHECKOUT\_SESSION\_ID}',

cancel\_url=url\_for('cancel', \_external=True),

)

return redirect(session.url, code=303)

except Exception as e:

return str(e)

@app.route('/success')

@login\_required

def success():

return "Subscription successful!"

@app.route('/cancel')

@login\_required

def cancel():

return "Subscription canceled."

**Create HTML Form for Selecting Membership and Handling Payments:**

**Create membership.html in the templates folder:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Membership - Artbox</title>

<link rel="stylesheet" href="{{ url\_for('static', filename='styles.css') }}">

<script src="https://js.stripe.com/v3/"></script>

</head>

<body>

<div class="container">

<h1>Choose Your Membership Plan</h1>

<form id="checkout-form" action="/create-checkout-session" method="POST">

<label>

<input type="radio" name="price\_id" value="PRICE\_ID\_FOR\_BASIC\_PLAN" required>

Basic - $9.99/month

</label><br>

<label>

<input type="radio" name="price\_id" value="PRICE\_ID\_FOR\_PREMIUM\_PLAN" required>

Premium - $39.99/month

</label><br>

<label>

<input type="radio" name="price\_id" value="PRICE\_ID\_FOR\_PRO\_PLAN" required>

Pro - $69.99/month

</label><br>

<button type="submit">Subscribe</button>

</form>

</div>

</body>

</html>

**Step 27: Update User Membership on Successful Payment**

**Set up the webhook endpoint:**

@app.route('/webhook', methods=['POST'])

def stripe\_webhook():

payload = request.get\_data(as\_text=True)

sig\_header = request.headers.get('Stripe-Signature')

event = None

try:

event = stripe.Webhook.construct\_event(

payload, sig\_header, 'YOUR\_STRIPE\_ENDPOINT\_SECRET'

)

except ValueError as e:

# Invalid payload

return str(e), 400

except stripe.error.SignatureVerificationError as e:

# Invalid signature

return str(e), 400

# Handle the event

if event['type'] == 'checkout.session.completed':

session = event['data']['object']

handle\_checkout\_session(session)

return '', 200

def handle\_checkout\_session(session):

customer\_email = session['customer\_email']

price\_id = session['display\_items'][0]['price']['id']

user = User.query.filter\_by(email=customer\_email).first()

if user:

if price\_id == 'PRICE\_ID\_FOR\_BASIC\_PLAN':

user.membership = 'Basic'

elif price\_id == 'PRICE\_ID\_FOR\_PREMIUM\_PLAN':

user.membership = 'Premium'

elif price\_id == 'PRICE\_ID\_FOR\_PRO\_PLAN':

user.membership = 'Pro'

db.session.commit()

**Update User model to include email:**

class User(UserMixin, db.Model):

id = db.Column(db.Integer, primary\_key=True)

username = db.Column(db.String(150), unique=True, nullable=False)

email = db.Column(db.String(150), unique=True, nullable=False)

password = db.Column(db.String(150), nullable=False)

membership = db.Column(db.String(50), nullable=False)

Run your application and expose the webhook endpoint. Make sure your endpoint is publicly accessible (you can use a service like ngrok during development).

**Step 28: Final Testing**

**Register new users and log in.**

**Subscribe to a membership plan and check the status update.**

**Generate AI images and verify limits based on the membership level.**

**Check profile to see the updated membership and image count.**

**Step 29: Refining the Front-End Design**

**Update styles.css**

body {

font-family: 'Helvetica Neue', Helvetica, Arial, sans-serif;

background-color: #f4f4f9;

color: #333;

margin: 0;

padding: 0;

}

.container {

width: 90%;

max-width: 800px;

margin: 50px auto;

background: #fff;

padding: 20px;

box-shadow: 0 0 10px rgba(0,0,0,0.1);

border-radius: 10px;

}

h1, h2, h3 {

color: #333;

margin-bottom: 20px;

}

form {

display: flex;

flex-direction: column;

}

label {

margin-bottom: 10px;

font-weight: bold;

}

input, select, button {

margin-bottom: 20px;

padding: 10px;

font-size: 16px;

border-radius: 5px;

border: 1px solid #ccc;

}

button {

background-color: #007bff;

color: white;

border: none;

cursor: pointer;

}

button:hover {

background-color: #0056b3;

}

nav {

background: #007bff;

color: white;

padding: 10px 0;

}

nav a {

color: white;

text-decoration: none;

margin: 0 15px;

font-weight: bold;

}

nav a:hover {

text-decoration: underline;

}

**Update HTML Templates**

**Update index.html:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Artbox</title>

<link rel="stylesheet" href="{{ url\_for('static', filename='styles.css') }}">

</head>

<body>

<nav>

<div class="container">

<a href="{{ url\_for('home') }}">Home</a>

{% if current\_user.is\_authenticated %}

<a href="{{ url\_for('profile') }}">Profile</a>

<a href="{{ url\_for('logout') }}">Logout</a>

{% else %}

<a href="{{ url\_for('login') }}">Login</a>

<a href="{{ url\_for('register') }}">Register</a>

{% endif %}

</div>

</nav>

<div class="container">

<h1>Welcome to Artbox</h1>

<p>Your ultimate destination for tattoo designs powered by AI.</p>

{% if current\_user.is\_authenticated %}

<h2>Generate a new tattoo design</h2>

<form id="generate-form" action="{{ url\_for('generate') }}" method="POST">

<label for="prompt">Describe your tattoo:</label>

<input type="text" id="prompt" name="prompt" required>

<button type="submit">Generate</button>

</form>

{% else %}

<p>Please <a href="{{ url\_for('login') }}">login</a> to generate tattoo designs.</p>

{% endif %}

</div>

<script>

document.getElementById('generate-form').addEventListener('submit', async function(e) {

e.preventDefault();

const prompt = document.getElementById('prompt').value;

const response = await fetch('/generate', {

method: 'POST',

headers: {

'Content-Type': 'application/json'

},

body: JSON.stringify({ prompt: prompt })

});

const result = await response.json();

if (response.ok) {

alert('Tattoo generated: ' + result.image\_url);

} else {

alert('Error: ' + result.error);

}

});

</script>

</body>

</html>

**Update login.html:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Login - Artbox</title>

<link rel="stylesheet" href="{{ url\_for('static', filename='styles.css') }}">

</head>

<body>

<div class="container">

<h1>Login</h1>

<form method="POST" action="{{ url\_for('login') }}">

<label for="username">Username:</label>

<input type="text" id="username" name="username" required>

<label for="password">Password:</label>

<input type="password" id="password" name="password" required>

<button type="submit">Login</button>

</form>

<p>Don't have an account? <a href="{{ url\_for('register') }}">Register</a></p>

</div>

</body>

</html>

**Update register.html:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Register - Artbox</title>

<link rel="stylesheet" href="{{ url\_for('static', filename='styles.css') }}">

</head>

<body>

<div class="container">

<h1>Register</h1>

<form method="POST" action="{{ url\_for('register') }}">

<label for="username">Username:</label>

<input type="text" id="username" name="username" required>

<label for="email">Email:</label>

<input type="email" id="email" name="email" required>

<label for="password">Password:</label>

<input type="password" id="password" name="password" required>

<label for="membership">Membership:</label>

<select id="membership" name="membership" required>

<option value="Basic">Basic</option>

<option value="Premium">Premium</option>

<option value="Pro">Pro</option>

</select>

<button type="submit">Register</button>

</form>

<p>Already have an account? <a href="{{ url\_for('login') }}">Login</a></p>

</div>

</body>

</html>

**Update profile.html:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Profile - Artbox</title>

<link rel="stylesheet" href="{{ url\_for('static', filename='styles.css') }}">

</head>

<body>

<div class="container">

<h1>Your Profile</h1>

<p>Membership: {{ current\_user.membership }}</p>

<p>Images Generated: {{ image\_count }} / {{ quota }}</p>

<a href="{{ url\_for('membership') }}">Change Membership</a>

</div>

</body>

</html>

**Step 30: Advanced Features and Scalability**

**Step 31: Continuous Integration and Deployment (CI/CD)**

**Step 1: Setting Up GitHub Repository**

**Create a new repository on GitHub and push your existing code to this repository.**

**Create a .github/workflows directory in the root of your project.**

**Step 2: Adding a CI/CD Pipeline with GitHub Actions**

**Create a workflow file in the .github/workflows directory, e.g., ci-cd.yml.**

name: CI/CD Pipeline

on:

push:

branches:

- main

pull\_request:

branches:

- main

jobs:

build:

runs-on: ubuntu-latest

steps:

- name: Checkout code

uses: actions/checkout@v2

- name: Set up Python

uses: actions/setup-python@v2

with:

python-version: '3.8'

- name: Install dependencies

run: |

python -m pip install --upgrade pip

pip install -r requirements.txt

- name: Run Tests

run: |

pytest

deploy:

needs: build

runs-on: ubuntu-latest

steps:

- name: Checkout code

uses: actions/checkout@v2

- name: Deploy to Heroku

env:

HEROKU\_API\_KEY: ${{ secrets.HEROKU\_API\_KEY }}

run: |

git remote add heroku https://git.heroku.com/<your-app-name>.git

git push heroku main

**Add secrets to GitHub repository:**

**Go to your GitHub repository settings.**

**Navigate to "Secrets" and add a new secret HEROKU\_API\_KEY with your Heroku API key.**

**Step 3: Setting Up Heroku for Deployment**

**Create a new application on Heroku.**

**Get the Heroku API key from the account settings.**

**Set up environment variables in Heroku (e.g., DATABASE\_URL, FLASK\_ENV, etc.).**

**Step 32: Monitoring and Logging**

***Step 1: Setting Up Logging***

***Use Python's built-in logging library to set up logging in your application.***

**Update app.py to configure logging:**

import logging

logging.basicConfig(level=logging.INFO)

@app.before\_request

def log\_request\_info():

logging.info(f"Request: {request.method} {request.url}")

logging.info(f"Body: {request.get\_data()}")

@app.after\_request

def log\_response\_info(response):

logging.info(f"Response: {response.status\_code}")

return response

***Step 2: Monitoring with New Relic or Prometheus***

***Sign up for a monitoring service like New Relic or Prometheus.***

***Install the New Relic Python agent or Prometheus client library.***

***For New Relic:***

pip install newrelic

**Configure New Relic:**

import newrelic.agent

newrelic.agent.initialize('newrelic.ini')

app = Flask(\_\_name\_\_)

newrelic.agent.register\_application()

**For Prometheus:**

pip install prometheus\_client

**Integrate Prometheus:**

from prometheus\_client import start\_http\_server, Summary

REQUEST\_TIME = Summary('request\_processing\_seconds', 'Time spent processing request')

@app.before\_first\_request

def start\_prometheus\_server():

start\_http\_server(8000)

**Step 33: User Feedback and Iterative Improvement**

**Add Feedback Model:**

class Feedback(db.Model):

id = db.Column(db.Integer, primary\_key=True)

user\_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)

image\_url = db.Column(db.String(255), nullable=False)

rating = db.Column(db.Integer, nullable=False)

comment = db.Column(db.String(255))

**Add Feedback Form in feedback.html:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Feedback - Artbox</title>

<link rel="stylesheet" href="{{ url\_for('static', filename='styles.css') }}">

</head>

<body>

<div class="container">

<h1>Provide Feedback</h1>

<form method="POST" action="{{ url\_for('feedback') }}">

<label for="image\_url">Image URL:</label>

<input type="text" id="image\_url" name="image\_url" required>

<label for="rating">Rating (1-5):</label>

<input type="number" id="rating" name="rating" min="1" max="5" required>

<label for="comment">Comment:</label>

<textarea id="comment" name="comment"></textarea>

<button type="submit">Submit Feedback</button>

</form>

</div>

</body>

</html>

**Handle Feedback Submission in app.py:**

@app.route('/feedback', methods=['GET', 'POST'])

@login\_required

def feedback():

if request.method == 'POST':

image\_url = request.form['image\_url']

rating = int(request.form['rating'])

comment = request.form.get('comment', '')

feedback = Feedback(user\_id=current\_user.id, image\_url=image\_url, rating=rating, comment=comment)

db.session.add(feedback)

db.session.commit()

return redirect(url\_for('profile'))

return render\_template('feedback.html')

**Step 34: Enhancing User Experience with Additional Features**

***Step 1: Implementing Tattoo Placement Visualization***

***Using a library like Three.js for 3D visualization or integrating with an AR framework can enable users to see how a tattoo looks on different body parts.***

**Update generate.html:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Visualize Tattoo - Artbox</title>

<link rel="stylesheet" href="{{ url\_for('static', filename='styles.css') }}">

<script src="https://cdnjs.cloudflare.com/ajax/libs/three.js/r128/three.min.js"></script>

<script src="https://cdn.jsdelivr.net/npm/@google/model-viewer/dist/model-viewer.min.js"></script>

</head>

<body>

<div class="container">

<h1>Visualize Your Tattoo</h1>

<div>

<model-viewer id="tattoo-viewer" src="{{ url\_for('static', filename='models/body.glb') }}" ar ar-modes="scene-viewer webxr quick-look" camera-controls auto-rotate></model-viewer>

</div>

<form id="upload-form" action="{{ url\_for('upload\_photo') }}" method="POST" enctype="multipart/form-data">

<label for="photo">Upload your photo:</label>

<input type="file" id="photo" name="photo" required>

<button type="submit">Upload and Visualize</button>

</form>

</div>

<script>

document.getElementById('upload-form').addEventListener('submit', async function(e) {

e.preventDefault();

const formData = new FormData();

formData.append('photo', document.getElementById('photo').files[0]);

const response = await fetch('/upload\_photo', {

method: 'POST',

body: formData

});

const result = await response.json();

if (response.ok) {

alert('Photo uploaded: ' + result.message);

} else {

alert('Error: ' + result.error);

}

});

</script>

</body>

</html>

**Handle Photo Upload in app.py:**

@app.route('/upload\_photo', methods=['POST'])

@login\_required

def upload\_photo():

if 'photo' not in request.files:

return jsonify({'error': 'No photo uploaded'}), 400

file = request.files['photo']

if file.filename == '':

return jsonify({'error': 'No photo selected'}), 400

if file:

filename = secure\_filename(file.filename)

filepath = os.path.join(app.config['UPLOAD\_FOLDER'], filename)

file.save(filepath)

# Process the photo to overlay the tattoo design

# This can involve image processing or AR integration

return jsonify({'message': 'Photo uploaded successfully'}), 200

return jsonify({'error': 'Failed to upload photo'}), 500

***Step 2: User Profiles and Portfolios***

***Enable users to create profiles and showcase their tattoo designs.***

***Update profile.html to include portfolio:***

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Profile - Artbox</title>

<link rel="stylesheet" href="{{ url\_for('static', filename='styles.css') }}">

</head>

<body>

<div class="container">

<h1>Your Profile</h1>

<p>Membership: {{ current\_user.membership }}</p>

<p>Images Generated: {{ image\_count }} / {{ quota }}</p>

<a href="{{ url\_for('membership') }}">Change Membership</a>

<h2>Your Portfolio</h2>

<div class="portfolio">

{% for image in portfolio\_images %}

<div class="portfolio-item">

<img src="{{ url\_for('static', filename='uploads/' + image.filename) }}" alt="Tattoo Image">

<p>{{ image.description }}</p>

</div>

{% endfor %}

</div>

</div>

</body>

</html>

**Add Portfolio Model:**

class PortfolioImage(db.Model):

id = db.Column(db.Integer, primary\_key=True)

user\_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)

filename = db.Column(db.String(255), nullable=False)

description = db.Column(db.String(255))

**Handle Portfolio Image Upload:**

@app.route('/upload\_portfolio', methods=['POST'])

@login\_required

def upload\_portfolio():

if 'file' not in request.files:

return redirect(request.url)

file = request.files['file']

if file.filename == '':

return redirect(request.url)

if file:

filename = secure\_filename(file.filename)

filepath = os.path.join(app.config['UPLOAD\_FOLDER'], filename)

file.save(filepath)

description = request.form.get('description', '')

portfolio\_image = PortfolioImage(user\_id=current\_user.id, filename=filename, description=description)

db.session.add(portfolio\_image)

db.session.commit()

return redirect(url\_for('profile'))

**Update profile.html form for uploading portfolio images:**

<form method="POST" action="{{ url\_for('upload\_portfolio') }}" enctype="multipart/form-data">

<label for="file">Upload Portfolio Image:</label>

<input type="file" id="file" name="file" required>

<label for="description">Description:</label>

<input type="text" id="description" name="description">

<button type="submit">Upload</button>

</form>

**Step 35: Implementing Advanced Search and Filters**

***Step 1: Update Database Models***

***Ensure the PortfolioImage model has attributes that can be used for searching and filtering.***

class PortfolioImage(db.Model):

id = db.Column(db.Integer, primary\_key=True)

user\_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)

filename = db.Column(db.String(255), nullable=False)

description = db.Column(db.String(255))

tags = db.Column(db.String(255)) # New field for tags

created\_at = db.Column(db.DateTime, default=datetime.utcnow) # New field for timestamp

***Step 2: Update Upload Form to Include Tags***

***Update the upload form in profile.html to allow users to add tags to their images.***

<form method="POST" action="{{ url\_for('upload\_portfolio') }}" enctype="multipart/form-data">

<label for="file">Upload Portfolio Image:</label>

<input type="file" id="file" name="file" required>

<label for="description">Description:</label>

<input type="text" id="description" name="description">

<label for="tags">Tags (comma-separated):</label>

<input type="text" id="tags" name="tags">

<button type="submit">Upload</button>

</form>

***Step 3: Handle Tags in Backend***

***Update the upload\_portfolio function to handle the new tags field.***

@app.route('/upload\_portfolio', methods=['POST'])

@login\_required

def upload\_portfolio():

if 'file' not in request.files:

return redirect(request.url)

file = request.files['file']

if file.filename == '':

return redirect(request.url)

if file:

filename = secure\_filename(file.filename)

filepath = os.path.join(app.config['UPLOAD\_FOLDER'], filename)

file.save(filepath)

description = request.form.get('description', '')

tags = request.form.get('tags', '')

portfolio\_image = PortfolioImage(user\_id=current\_user.id, filename=filename, description=description, tags=tags)

db.session.add(portfolio\_image)

db.session.commit()

return redirect(url\_for('profile'))

***Step 4: Implement Search and Filter Functionality***

***Add a new route and template for the search functionality.***

**Update app.py:**

@app.route('/search', methods=['GET', 'POST'])

@login\_required

def search():

query = request.args.get('query')

tags = request.args.get('tags')

if query or tags:

results = PortfolioImage.query.filter(

or\_(

PortfolioImage.description.ilike(f'%{query}%'),

PortfolioImage.tags.ilike(f'%{tags}%')

)

).all()

else:

results = []

return render\_template('search.html', results=results)

**Create search.html:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Search - Artbox</title>

<link rel="stylesheet" href="{{ url\_for('static', filename='styles.css') }}">

</head>

<body>

<div class="container">

<h1>Search Portfolio</h1>

<form method="GET" action="{{ url\_for('search') }}">

<label for="query">Search by Description:</label>

<input type="text" id="query" name="query">

<label for="tags">Search by Tags (comma-separated):</label>

<input type="text" id="tags" name="tags">

<button type="submit">Search</button>

</form>

<div class="results">

{% for image in results %}

<div class="result-item">

<img src="{{ url\_for('static', filename='uploads/' + image.filename) }}" alt="Tattoo Image">

<p>{{ image.description }}</p>

<p>Tags: {{ image.tags }}</p>

</div>

{% endfor %}

</div>

</div>

</body>

</html>

**Step 36: Adding Social Features**

**Integrate social features to allow users to interact with each other’s designs.**

***Step 1: Update Database Models***

***Create models for likes and comments.***

class Like(db.Model):

id = db.Column(db.Integer, primary\_key=True)

user\_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)

image\_id = db.Column(db.Integer, db.ForeignKey('portfolio\_image.id'), nullable=False)

class Comment(db.Model):

id = db.Column(db.Integer, primary\_key=True)

user\_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)

image\_id = db.Column(db.Integer, db.ForeignKey('portfolio\_image.id'), nullable=False)

content = db.Column(db.String(255), nullable=False)

created\_at = db.Column(db.DateTime, default=datetime.utcnow)

***Step 2: Implement Like and Comment Functionality***

***Add routes to handle likes and comments.***

**Update app.py:**

@app.route('/like/<int:image\_id>', methods=['POST'])

@login\_required

def like(image\_id):

like = Like(user\_id=current\_user.id, image\_id=image\_id)

db.session.add(like)

db.session.commit()

return redirect(request.referrer)

@app.route('/comment/<int:image\_id>', methods=['POST'])

@login\_required

def comment(image\_id):

content = request.form['content']

comment = Comment(user\_id=current\_user.id, image\_id=image\_id, content=content)

db.session.add(comment)

db.session.commit()

return redirect(request.referrer)

***Step 3: Update Templates to Show Likes and Comments***

***Update profile.html and search.html to include like and comment forms.***

<div class="result-item">

<img src="{{ url\_for('static', filename='uploads/' + image.filename) }}" alt="Tattoo Image">

<p>{{ image.description }}</p>

<p>Tags: {{ image.tags }}</p>

<form method="POST" action="{{ url\_for('like', image\_id=image.id) }}">

<button type="submit">Like</button>

</form>

<form method="POST" action="{{ url\_for('comment', image\_id=image.id) }}">

<input type="text" name="content" placeholder="Add a comment">

<button type="submit">Comment</button>

</form>

<div class="comments">

{% for comment in image.comments %}

<p>{{ comment.content }} - {{ comment.user.username }}</p>

{% endfor %}

</div>

</div>

**Step 37: Deploying the Application**

***Step 1: Deploy to Heroku***

***Ensure your Procfile, requirements.txt, and other necessary deployment files are correctly set up.***

**Example Procfile:**

web: gunicorn app:app

**Deploy to Heroku:**

Commit your changes.

**Push to Heroku:**

git push heroku main

***Step 2: Configure Environment Variables***

***Set environment variables on Heroku for production.***

heroku config:set FLASK\_ENV=production

heroku config:set DATABASE\_URL=<your-database-url>

**Step 38: Final Testing and Launch**

**Step 39: Continuous Improvement and Maintenance**

**Step 40: Version Control and Release Management**

**Step 41: Monitoring and Analytics**

**Step 42: Adaptation to Emerging Technologies**