MindMash.AI - codebase

Dashboard.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Dashboard - MindMash.AI</title>
    <link rel="stylesheet" href="/static/style.css">
    <link
href="https://fonts.googleapis.com/css2?family=Orbitron:wght@400;700&displ
ay=swap" rel="stylesheet">
    <style>
        .loading {
            display: none;
            font-size: 0.9em;
            color: #ff00ff;
            margin-left: 10px;
            font-weight: bold;
            text-shadow: 0 0 10px #ff00ff;
        }

        .chat-box {
            width: 100%;
            height: 400px;
            overflow-y: auto;
            background: rgba(0, 0, 0, 0.6);
            border: 2px solid #ff00ff;
            border-radius: 10px;
            padding: 15px;
            color: #ffffff;
            font-family: 'Orbitron', sans-serif;
        }

        .chat-message {
            margin-bottom: 10px;
            line-height: 1.4;
        }
        .chat-message .speaker {
            font-weight: bold;
            color: #ffffff; /* AI/User names in white */
        }

        /* Color-coded AI responses */
        .chat-message.ChatGPT .message-content { color: #eecdf7; }
        .chat-message.Grok .message-content { color: #c2f3d8; }
        .chat-message.Gemini .message-content { color: #f6f7b9; }
        .chat-message.user .message-content { color: #00ffff; }

    </style>
</head>
<body>
    <div class="container">
        <header class="header">
            <h1 class="title">MindMash.AI <span class="testing">(Testing
Mode)</span></h1>
            <p class="tagline">Where Minds and Machines Collide</p>
        </header>

        <h1 class="welcome-title">Welcome, {{ display_name if display_name
else username }}!</h1>

        <div class="dashboard-grid">
            <div class="chat-section">
                <h2 class="section-title">
                    Chat:
                    <span class="loading" id="loading">Gathering
MINDS...</span>
                </h2>
                <div class="chat-box" id="chatBox">
                    <!-- Chat messages appear here -->
                </div>
                <div class="chat-input">
                    <span class="tooltip">
                        <span class="tooltiptext">Choose an AI or 'All
AIs' for collaboration, then type your message and press 'Chat'</span>
                    </span>
                    <select id="aiSelect" class="chat-input-field">
```

```
                        <option value="All">All AIs
(Collaborative)</option>
                        <option value="Grok">Grok</option>
                        <option value="ChatGPT">ChatGPT</option>
                        <option value="Gemini">Gemini</option>
                    </select>
                    <input type="text" id="messageInput"
placeholder="Enter your message..." class="chat-input-field">
                    <input type="number" id="turnsInput" min="1" max="10"
value="1" class="chat-turns" title="Number of AI responses per chat (max
10)">
                    <button id="sendChat" class="button
chat-button">Chat</button>
                </div>
            </div>

            <div class="mode-section">
                <h2 class="section-title">Modes</h2>
                <div class="mode basic">
                    <h3>Basic Mode (Free)</h3>
                    <p>Chat with Grok, ChatGPT, and Gemini.</p>
                </div>
                <div class="mode premium">
                    <h3>Premium Mode {% if not is_premium and beta_mode
%}(Upgrade - Beta Unavailable){% elif not is_premium %}(Upgrade){% endif
%}</h3>
                    <p>Unlock AI debates, collaboration spaces, and
more!</p>
                    {% if not is_premium and not beta_mode %}
                        <a href="/premium" class="button glow">Upgrade</a>
                    {% elif not is_premium and beta_mode %}
                        <a href="/premium" class="button glow"
disabled>Upgrade</a>
                    {% endif %}
                </div>
                <div class="nav-buttons">
                    <a href="/logout" class="button">Logout</a>
                    <a href="/profile" class="button">Profile</a>
                    <a href="/feedback" class="button">Feedback</a>
                </div>
            </div>
        </div>
    </div>

    <script>
        const chatBox = document.getElementById('chatBox');
        const loadingIndicator = document.getElementById('loading');
        const sendChatButton = document.getElementById('sendChat');
        const displayName = "{{ display_name if display_name else username
}}";

        function appendMessage(speaker, message) {
            const messageDiv = document.createElement('div');
            messageDiv.className = `chat-message ${speaker}`;
            messageDiv.innerHTML = `
                <span class="speaker">${speaker}:</span>
                <span class="message-content">${message}</span>
            `;
            chatBox.appendChild(messageDiv);
            chatBox.scrollTop = chatBox.scrollHeight;
        }

        sendChatButton.addEventListener('click', async () => {
            const userMessage =
document.getElementById('messageInput').value.trim();
            const selectedAI = document.getElementById('aiSelect').value;
            const turns = document.getElementById('turnsInput').value;

            if (!userMessage) return;

            appendMessage(displayName, userMessage); // Use actual
username
            loadingIndicator.style.display = 'inline';

            try {
                const response = await fetch('/chat', {
                    method: 'POST',
                    headers: { 'Content-Type': 'application/json' },
                    body: JSON.stringify({
                        message: userMessage,
                        selected_ai: selectedAI,
                        turns: parseInt(turns)
                    })
                });

                const data = await response.json();
                if (data.history) {
                    data.history.slice(-turns).forEach(entry => {
                        appendMessage(entry.speaker, entry.content);
                    });
                }
            } catch (err) {
                appendMessage('System', 'Error fetching AI response.');
            } finally {
                loadingIndicator.style.display = 'none';
                document.getElementById('messageInput').value = '';
            }
        });
    </script>
</body>
</html>
```

edit profile.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Edit Profile - MindMash.AI</title>
    <link rel="stylesheet" href="/static/style.css">
    <link
href="https://fonts.googleapis.com/css2?family=Orbitron:wght@400;700&displ
ay=swap" rel="stylesheet">
</head>
<body>
    <div class="container">
        <h1 class="title">Edit Profile</h1>
        {% with messages = get_flashed_messages() %}
            {% if messages %}
```

```
                {% for message in messages %}
                    <p>{{ message }}</p>
                {% endfor %}
            {% endif %}
        {% endwith %}
        <form method="POST">
            <label for="display_name">Display Name:</label>
            <input type="text" id="display_name" name="display_name"
value="{{ display_name }}" required class="chat-input-field">
            <button type="submit" class="button glow">Save</button>
            <a href="{{ url_for('profile') }}" class="button">Back to
Profile</a>
        </form>
    </div>
</body>
</html>
```

feedback.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Feedback - MindMash.AI</title>
    <link rel="stylesheet" href="/static/style.css">
    <link
href="https://fonts.googleapis.com/css2?family=Orbitron:wght@400;700&displ
ay=swap" rel="stylesheet">
    <style>
        .large-textarea {
            width: 100%;
            max-width: 600px;
            min-height: 200px; /* Increased height for more space */
            font-size: 1.1em; /* Slightly larger text for readability */
            padding: 10px;
            margin-bottom: 15px;
            resize: vertical; /* Allow vertical resizing */
        }
    </style>
```

```html
</head>
<body>
    <div class="container">
        <h1 class="title">Feedback</h1>
        <p>Help us improve MindMash.AI! Share your thoughts, {{ username
}}.</p>
        {% with messages = get_flashed_messages() %}
            {% if messages %}
                {% for message in messages %}
                    <p>{{ message }}</p>
                {% endfor %}
            {% endif %}
        {% endwith %}
        <form method="POST">
            <textarea name="feedback" class="large-textarea"
placeholder="Enter your feedback..." required></textarea>
            <button type="submit" class="button glow">Submit</button>
            <a href="{{ url_for('dashboard') }}" class="button">Back to
Dashboard</a>
        </form>
    </div>
</body>
</html>
```

index.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>MindMash.AI: Where Minds and Machines Collide</title>
    <link rel="stylesheet" href="/static/style.css">
</head>
<body>
    <div class="container">
        <h1>MindMash.AI <span class="testing">(Testing Mode)</span></h1>
        <p class="tagline">Where Minds and Machines Collide</p>
        <div class="game-info">Points: <span id="points">{{ points
}}</span></div>
```

```
        <div class="chat-box" id="chatBox"></div>
        <div class="input-area">
            <input type="text" id="messageInput" placeholder="Enter your message..." autocomplete="off">
            <button id="sendChat">Chat</button>
            <select id="betAI">
                <option value="Grok">Bet on Grok</option>
                <option value="ChatGPT">Bet on ChatGPT</option>
                <option value="Gemini">Bet on Gemini</option>
            </select>
            <input type="number" id="betPoints" placeholder="Points to bet" min="1" max="100">
            <button id="startDebate">Start Debate</button>
        </div>
    </div>
    <script src="/static/script.js"></script>
</body>
</html>
```

landing.html

```
<head>
    <meta charset="UTF-8">
    <title>MindMash.AI (Testing Mode)</title>
    <link rel="stylesheet" href="/static/style.css">
    <link href="https://fonts.googleapis.com/css2?family=Orbitron:wght@400;700&display=swap" rel="stylesheet">
</head>
<body>
    <div class="container">
        <header class="header">
            <h1 class="title">MindMash.AI <span class="testing">(Testing Mode)</span></h1>
            <p class="tagline">Where Minds and Machines Collide</p>
        </header>

        <div class="intro">
            <p>Chat, debate, and collaborate with AI minds—Grok, ChatGPT, and Gemini</p>
```

```html
            <p>Unlock Premium for advanced AI debates, collaboration spaces, and more!</p>
            <a href="{{ url_for('login') }}?signup=1" class="button">Sign Up</a>
            <a href="{{ url_for('login') }}" class="button">Sign In</a>
        </div>

        <div class="demo">
            <img src="/static/ai_chat.gif" alt="AI Chat Demo" class="demo-gif">
        </div>
    </div>

    <footer class="footer">
        <p>
            © 2025
            <a href="{{ url_for('privacy') }}" class="footer-link">MindMash.AI</a> |
            <a href="{{ url_for('privacy') }}" class="footer-link">Privacy Policy</a> |
            <a href="{{ url_for('terms') }}" class="footer-link">Terms of Service</a>
        </p>
    </footer>
</body>
</html>
```

premium.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Profile - MindMash.AI</title>
    <link rel="stylesheet" href="/static/style.css">
</head>
<body>
    <div class="container">
        <h1>Profile</h1>
        {% with messages = get_flashed_messages() %}
```

```
            {% if messages %}
                <p class="flash">{{ messages[0] }}</p>
            {% endwith %}
        <p>Username: {{ username }}</p>
        <p>Subscription: {{ "Premium" if is_premium else "Basic" }}</p>
        <a href="/dashboard" class="button">Back to Dashboard</a>
    </div>
</body>
</html>
```

privacy.html

```
!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Privacy Policy - MindMash.AI</title>
    <link rel="stylesheet" href="/static/style.css">
    <link
href="https://fonts.googleapis.com/css2?family=Orbitron:wght@400;700&displ
ay=swap" rel="stylesheet">
</head>
<body>
    <div class="container">
        <h1 class="title">Privacy Policy</h1>
        <p>MindMash.AI is currently in beta testing for Basic Mode only.
We collect minimal user data for chat functionality—email and display
name—for authentication and interaction. Data is stored securely in a
SQLite database and not shared with third parties. During beta, we may log
usage for improvement purposes, but no personal data is sold or
exposed.</p>
        <a href="/dashboard" class="button">Back to Dashboard</a>
    </div>
</body>
</html>
```

profile.html

```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
    <meta charset="UTF-8">
    <title>Profile - MindMash.AI</title>
    <link rel="stylesheet" href="/static/style.css">
</head>
<body>
    <div class="container">
        <h1 class="title">Profile</h1>
        {% with messages = get_flashed_messages() %}
            {% if messages %}
                <p class="flash">{{ messages[0] }}</p>
            {% endif %}
        {% endwith %}

        <div class="profile-details">
            <p><strong>Display Name:</strong> {{ display_name if
display_name else username }}</p>
            <p><strong>Username:</strong> {{ username }}</p>
            <p><strong>Subscription:</strong> {{ "Premium" if is_premium
else "Basic" }}</p>
            <p><strong>Joined:</strong> {{ "February 22, 2025" }} (mock
date—can fetch from DB later)</p>
        </div>

        <div class="profile-actions">
            <a href="/profile/edit" class="button glow">Edit Profile</a>
            <a href="/dashboard" class="button">Back to Dashboard</a>
        </div>
    </div>
</body>
</html>
```

set_name.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Set Display Name - MindMash.AI</title>
    <link rel="stylesheet" href="/static/style.css">
```

```
</head>
<body>
    <div class="container">
        <h1>Set Your Display Name</h1>
        <p>Welcome, {{ email }}! Choose a display name for
MindMash.AI.</p>
        <form method="POST" action="/set-name">
            <input type="hidden" name="email" value="{{ email }}">
            <input type="hidden" name="google_id" value="{{ google_id }}">
            <input type="text" name="display_name" value="{{ display_name
}}" placeholder="Display Name" required>
            <button type="submit" class="button glow">Save</button>
        </form>
        <a href="/login" class="button">Back to Sign In</a>
    </div>
</body>
</html>
```

terms.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Terms of Service - MindMash.AI</title>
    <link rel="stylesheet" href="/static/style.css">
    <link
href="https://fonts.googleapis.com/css2?family=Orbitron:wght@400;700&displ
ay=swap" rel="stylesheet">
</head>
<body>
    <div class="container">
        <h1 class="title">Terms of Service</h1>
        <p>By using MindMash.AI during beta testing, you agree to use
Basic Mode features (chat with Grok, ChatGPT, and Gemini) responsibly.
Premium features are unavailable during beta. We reserve the right to
modify or terminate beta access at any time. Feedback is encouraged and
appreciated for improving the service.</p>
        <a href="/dashboard" class="button">Back to Dashboard</a>
    </div>
```

```
</body>
</html>
```

app.py

```
import sqlite3
import os
import requests
from flask import Flask, render_template, request, session, redirect,
url_for, jsonify, flash
from authlib.integrations.flask_client import OAuth
from dotenv import load_dotenv
import logging
import google.generativeai as genai
import openai
# import stripe  # Uncomment if needed

# Load environment variables (optional, since we're hardcoding now)
load_dotenv()

# -------------------------
# ✅ Configure logging
# -------------------------
logging.basicConfig(level=logging.INFO, format='%(asctime)s -
%(levelname)s - %(message)s')
logger = logging.getLogger(__name__)

# -------------------------
# ✅ Initialize Flask app
# -------------------------
app = Flask(__name__)
app.secret_key = "your-secret-key-here"  # Replace with a secure value in
production

# -------------------------
# ✅ Beta mode flag
# -------------------------
beta_mode = True

# -------------------------
# ✅ Hardcoded OAuth Credentials
# ------------------------
CLIENT_ID =
"237809198690-ibrgillsl3n0s2909c40m7bn5ujs1hk2.apps.googleusercontent.com"
CLIENT_SECRET = "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"

# ✅ Log to verify correct loading
logger.info(f"Client ID: {CLIENT_ID}")
logger.info(f"Client Secret: {CLIENT_SECRET}")

# ------------------------
# ✅ Initialize OAuth
# ------------------------
oauth = OAuth(app)

# ✅ Register Google OAuth with correct variables
oauth.register(
    name="google",
    client_id=CLIENT_ID,  # Correct reference
    client_secret=CLIENT_SECRET,  # Correct reference

server_metadata_url="https://accounts.google.com/.well-known/openid-config
uration",
    client_kwargs={"scope": "openid email profile"}
)

# ------------------------
# ✅ Hardcoded API Keys
# ------------------------
XAI_API_KEY =
"xai-ozgjayVZHRPwiXJroH1mU3GGdRr7HQoEJOWea9hIcsLuTMmo3R7nxWzNLT88AtD5bxJRP
ZyDrxkvHofY"
OPENAI_API_KEY =
"sk-proj-WH8rWSaTAAhV-qjsMgVhSWkoZ6MO8uPDxdlbOr08yqUWJuivOZorLJWT6g5pnD-xs
bEcOg2dJMT3BlbkFJfBCOtQs3kC8xtvp0ca1Ghco9bMs1l-oEsa-HQd5z1KT1reEYRqLO6Oy2q
JF-QYiKt9x8CrnqkA"
GEMINI_API_KEY = "AIzaSyDoYRxV8T2TIXdxDKY4z_bVzeVaSkzyL3k"
STRIPE_SECRET_KEY = "your-stripe-secret-key"
STRIPE_PUBLISHABLE_KEY = "your-stripe-publishable-key"
```

```
# -------------------------
# ✅ API URLs and configurations
# -------------------------
XAI_API_URL = "https://api.x.ai/v1/chat/completions"

# -------------------------
# ✅ Configure APIs
# -------------------------
openai.api_key = OPENAI_API_KEY
# stripe.api_key = STRIPE_SECRET_KEY  # Uncomment if using Stripe
genai.configure(api_key=GEMINI_API_KEY)

# Enhanced system prompts for AIs
system_prompts = {
    "Grok": (
        "You are Grok, an AI created by xAI with a cosmic perspective,
offering humor and deep insights. "
        "When responding to a single user message, address the user
directly, offering an engaging perspective. "
        "In collaborative sessions, build on the ideas of ChatGPT and
Gemini—explore alternate dimensions of thought, "
        "inject humor where appropriate, and challenge assumptions to
broaden understanding. "
        "Embrace curiosity and cosmic wonder as you co-create with the
others."
    ),
    "ChatGPT": (
        "You are ChatGPT, a thoughtful and engaging AI. When responding to
a single user message, address the user directly, "
        "providing clarity and context. In collaborative sessions with
Grok and Gemini, act as a synthesizer—connecting diverse viewpoints, "
        "highlighting contrasts, and offering grounded insights. Aim to
keep the conversation constructive, curious, and fluid. "
        "Encourage further exploration and thoughtful reflection."
    ),
    "Gemini": (
        "You are Gemini, an AI with forward-thinking insights and a
passion for exploration. "
        "When responding to a single user message, address the user
directly and provide innovative perspectives. "
```

```
        "In collaborative sessions with Grok and ChatGPT, expand upon
their responses by offering novel ideas or futuristic concepts. "
        "Encourage multidimensional thinking and challenge both Grok and
ChatGPT to explore new possibilities. "
        "Maintain a balance between vision and practicality."
    )
}

# Global conversation history
conversation_history = []

# Database connection and initialization
def get_db_connection():
    conn = sqlite3.connect("users.db", timeout=30)
    conn.row_factory = sqlite3.Row
    return conn

def init_db():
    with get_db_connection() as conn:
        conn.execute("""
            CREATE TABLE IF NOT EXISTS users (
                username TEXT PRIMARY KEY,
                display_name TEXT,
                google_id TEXT UNIQUE,
                is_premium INTEGER DEFAULT 0
            )
        """)
        conn.commit()

init_db()

# ------------------------
# ✅ AI Response Functions
# ------------------------

def get_grok_response(history):
    """Get response from Grok (xAI) API."""
    try:
        messages = [{"role": "system", "content": system_prompts["Grok"]}]
+ [
            {"role": "assistant" if msg["speaker"] in ["Grok", "ChatGPT",
"Gemini"] else "user",
             "content": f"{msg['speaker']}: {msg['content']}"}
            for msg in history
        ]

        headers = {
            "Authorization": f"Bearer {XAI_API_KEY}",
            "Content-Type": "application/json"
        }
        payload = {"model": "grok-beta", "messages": messages}

        response = requests.post(XAI_API_URL, headers=headers,
json=payload, timeout=30)
        response.raise_for_status()  # Raises an HTTPError for bad
responses

        response_data = response.json()

        # ✅ Check response structure and handle gracefully
        if "choices" in response_data and response_data["choices"]:
            raw_message =
response_data["choices"][0]["message"]["content"]
            return raw_message.replace("Grok:", "").strip() if raw_message
else "No response from Grok."
        else:
            logger.warning("Grok API response missing 'choices'.")
            return "Oops! Grok didn't respond properly."

    except requests.exceptions.HTTPError as http_err:
        logger.error(f"HTTP error with Grok: {http_err}")
    except Exception as e:
        logger.error(f"Grok response error: {e}")
    return "Oops! We couldn’t connect to Grok—please try again."


def get_chatgpt_response(history):
    try:
        # Construct the messages for the ChatGPT API call
        messages = [{"role": "system", "content":
system_prompts["ChatGPT"]}]
        for msg in history:
            role = "assistant" if msg["speaker"] in ["Grok", "ChatGPT",
"Gemini"] else "user"
            messages.append({"role": role, "content": f"{msg['speaker']}:
{msg['content']}"})

        # Use the updated OpenAI ChatCompletion call
        response = openai.ChatCompletion.create(
            model="gpt-3.5-turbo",
            messages=messages,
            timeout=30
        )

        # Extract the assistant's message
        raw_message = response["choices"][0]["message"]["content"].strip()

        # Remove redundant prefixes if present
        if raw_message.startswith("ChatGPT: "):
            raw_message = raw_message[len("ChatGPT: "):]

        return raw_message

    except Exception as e:
        logger.error(f"ChatGPT response error: {e}")
        return "Oops! We couldn’t connect to ChatGPT—please try again."

def get_gemini_response(history):
    """Get response from Gemini (Google Generative AI) API."""
    try:
        model = genai.GenerativeModel("gemini-pro")
        prompt = "\n".join(f"{msg['speaker']}: {msg['content']}" for msg
in history)
        response =
model.generate_content(f"{system_prompts['Gemini']}\n\n{prompt}")

        if response and hasattr(response, 'text') and response.text:
            return response.text.replace("Gemini:", "").strip()
        else:
            logger.warning("Gemini returned no text.")
            return "Oops! Gemini didn't provide a response."

    except Exception as e:
        logger.error(f"Gemini response error: {e}")
    return "Oops! We couldn’t connect to Gemini—please try again."

# ------------------------
# ✅ Routes
# ------------------------

@app.route("/")
def landing():
    """Render the landing page."""
    return render_template("landing.html")


@app.route("/login")
def login():
    """Initiate Google OAuth login."""
    # ✅ Use the correctly defined variable: CLIENT_ID (or set
GOOGLE_CLIENT_ID directly)
    logger.info(f"Starting login with Client ID: {CLIENT_ID}")

    if "username" in session:
        return redirect(url_for("dashboard"))

    redirect_uri = url_for("google_callback", _external=True)
    logger.info(f"Redirect URI: {redirect_uri}")

    return oauth.google.authorize_redirect(redirect_uri)

@app.route("/auth/google/callback")
def google_callback():
    """Handle Google OAuth callback and user session setup."""
    logger.info("Entered callback")
    try:
        token = oauth.google.authorize_access_token()
        if not token:
            logger.error("Login failed: No token received")
            flash("Authentication failed!")
            return redirect(url_for("landing"))

        session["google_token"] = token["access_token"]
        user_info = token["userinfo"]
        google_id = user_info["sub"]
        email = user_info["email"]
        display_name = user_info.get("name", email.split("@")[0])

        with get_db_connection() as db:
            user = db.execute("SELECT * FROM users WHERE google_id = ?",
(google_id,)).fetchone()
            if user:
                session["username"] = user["username"]
                logger.info(f"Existing user logged in: {email}")
                flash("Welcome back to MindMash.AI!")
            else:
                db.execute(
                    "INSERT INTO users (username, display_name, google_id,
is_premium) VALUES (?, ?, ?, ?)",
                    (email, display_name, google_id, 0 if beta_mode else
0)
                )
                db.commit()
                session["username"] = email
                logger.info(f"New user signed up: {email}")
                flash("Welcome to MindMash.AI!")

        return redirect(url_for("dashboard"))
    except Exception as e:
        logger.error(f"Callback error: {str(e)}")
        flash(f"Login error: {str(e)}")
        return redirect(url_for("landing"))

@app.route("/logout")
def logout():
    """Log out the user by clearing the session."""
    session.pop("username", None)
    session.pop("google_token", None)
    logger.info("User logged out")
```

```python
    flash("You have been logged out.")
    return redirect(url_for("landing"))

@app.route("/dashboard")
def dashboard():
    """Render the main dashboard (template assumed to exist)."""
    if "username" not in session:
        return redirect(url_for("login"))
    with get_db_connection() as db:
        user = db.execute("SELECT username, display_name, is_premium FROM
users WHERE username = ?", (session["username"],)).fetchone()
    return render_template("dashboard.html", username=user["username"],
display_name=user["display_name"], is_premium=user["is_premium"],
beta_mode=beta_mode)

@app.route("/chat", methods=["POST"])
def chat():
    """Handle chat messages and return AI responses."""
    logger.info(f"Chat request from {session.get('username',
'Anonymous')}")
    if "username" not in session:
        return jsonify({"error": "Unauthorized"}), 401

    data = request.json
    user_message = data.get("message")
    selected_ai = data.get("selected_ai", "All")
    num_turns = data.get("turns", 1)  # Limit to 1 turn per AI for clean
flow

    if not user_message:
        return jsonify({"error": "No message provided"}), 400

    # Get user's display_name from session or database
    with get_db_connection() as db:
        display_name = db.execute("SELECT display_name FROM users WHERE
username = ?", (session["username"],)).fetchone()["display_name"]

    # Append human message (only from prompt, no errors)
    conversation_history.append({"speaker": display_name, "content":
user_message})
```

```
    # Handle single AI or all AIs for collaboration
    if selected_ai in ["Grok", "ChatGPT", "Gemini"]:
        # Single AI response, targeting only the user's last message
        message = None
        try:
            # Get only the user's last message for the single AI response
            user_history = [msg for msg in conversation_history if
msg["speaker"] == display_name][-1:]
            if user_history:
                if selected_ai == "Grok":
                    message = get_grok_response(user_history)
                elif selected_ai == "ChatGPT":
                    message = get_chatgpt_response(user_history)
                elif selected_ai == "Gemini":
                    message = get_gemini_response(user_history)

            if message and message.strip():
                conversation_history.append({"speaker": selected_ai,
"content": message})
            else:
                logger.warning(f"No valid response from {selected_ai}")
                conversation_history.append({"speaker": selected_ai,
"content": "Oops! We couldn’t connect to this AI—please try again"})
        except Exception as e:
            logger.error(f"Error from {selected_ai}: {e}")
            conversation_history.append({"speaker": selected_ai,
"content": "Oops! We couldn’t connect to this AI—please try again"})
    else:  # "All" for collaborative session
        # Define AI order for collaborative, non-repetitive responses
        ai_participants = ["Grok", "ChatGPT", "Gemini"]
        current_index = len(conversation_history) % len(ai_participants)

        # Generate one response per AI turn, ensuring collaboration and no
repetition
        for _ in range(num_turns):
            current_ai = ai_participants[current_index]
            message = None
            try:
                history_text = "\n".join([f"{msg['speaker']}:
{msg['content']}" for msg in conversation_history])
                if current_ai == "Grok":
                    message = get_grok_response(conversation_history)
                    if message and any(prev_msg["speaker"] == "Grok" and
prev_msg["content"] in message for prev_msg in conversation_history[-3:]):
                        message = f"Building on my earlier insight,
{message}"
                elif current_ai == "ChatGPT":
                    message = get_chatgpt_response(conversation_history)
                    if message and any(prev_msg["speaker"] == "ChatGPT"
and prev_msg["content"] in message for prev_msg in
conversation_history[-3:]):
                        message = f"Continuing our discussion, {message}"
                elif current_ai == "Gemini":
                    message = get_gemini_response(conversation_history)
                    if message and any(prev_msg["speaker"] == "Gemini" and
prev_msg["content"] in message for prev_msg in conversation_history[-3:]):
                        message = f"Adding to our collective wisdom,
{message}"

                if message and message.strip():
                    conversation_history.append({"speaker": current_ai,
"content": message})
                else:
                    logger.warning(f"No valid response from {current_ai}")
                    conversation_history.append({"speaker": current_ai,
"content": "Oops! We couldn’t connect to this AI—please try again"})
            except Exception as e:
                logger.error(f"Error from {current_ai}: {e}")
                conversation_history.append({"speaker": current_ai,
"content": "Oops! We couldn’t connect to this AI—please try again"})

            current_index = (current_index + 1) % len(ai_participants)

    return jsonify({"history": conversation_history})

@app.route("/profile")
def profile():
    """Display the user's profile information."""
```

```python
    if "username" not in session:
        return redirect(url_for("login"))
    with get_db_connection() as db:
        user = db.execute("SELECT username, display_name, is_premium FROM 
users WHERE username = ?", (session["username"],)).fetchone()
    return render_template("profile.html", username=user["username"], 
display_name=user["display_name"], is_premium=user["is_premium"], 
beta_mode=beta_mode)

@app.route("/profile/edit", methods=["GET", "POST"])
def edit_profile():
    """Allow the user to edit their display name."""
    if "username" not in session:
        return redirect(url_for("login"))

    if request.method == "POST":
        display_name = request.form.get("display_name")
        if display_name:
            with get_db_connection() as db:
                db.execute(
                    "UPDATE users SET display_name = ? WHERE username = 
?",
                    (display_name, session["username"])
                )
                db.commit()
            logger.info(f"User {session['username']} updated display_name 
to {display_name}")
            flash("Display name updated successfully!")
        return redirect(url_for("profile"))

    with get_db_connection() as db:
        user = db.execute("SELECT display_name FROM users WHERE username = 
?", (session["username"],)).fetchone()
    return render_template("edit_profile.html", 
display_name=user["display_name"] or "", beta_mode=beta_mode)

@app.route("/premium", methods=["GET", "POST"])
def premium():
    """Handle premium upgrade (disabled during beta)."""
    if "username" not in session:
        return redirect(url_for("login"))

    with get_db_connection() as db:
        user = db.execute("SELECT is_premium FROM users WHERE username =
?", (session["username"],)).fetchone()
        is_premium = user["is_premium"] if user else 0

    if is_premium:
        flash("You are already on Premium!")
        return redirect(url_for("dashboard"))

    if beta_mode:
        flash("Premium Mode is not available during beta testing. Enjoy
Basic Mode!")
        return redirect(url_for("dashboard"))

    if request.method == "POST":
        try:
            with get_db_connection() as db:
                db.execute("UPDATE users SET is_premium = 1 WHERE username
= ?", (session["username"],))
                db.commit()
            flash("Upgraded to Premium Mode!")
            return redirect(url_for("dashboard"))
        except Exception as e:
            logger.error(f"Premium upgrade error: {e}")
            flash(f"Error upgrading to Premium: {str(e)}")
            return redirect(url_for("dashboard"))

    return render_template("premium.html",
stripe_publishable_key=STRIPE_PUBLISHABLE_KEY, beta_mode=beta_mode)

@app.route("/feedback", methods=["GET", "POST"])
def feedback():
    """Handle user feedback during beta."""
    if request.method == "POST":
        feedback = request.form.get("feedback")
        if feedback:
            logger.info(f"Feedback from {session.get('username',
'Anonymous')}: {feedback}")
```

```
            flash("Thank you for your feedback!")
        return redirect(url_for("dashboard"))
    return render_template("feedback.html",
username=session.get("username", "Guest"), beta_mode=beta_mode)

@app.route("/privacy")
def privacy():
    """Render the privacy policy page."""
    return render_template("privacy.html", beta_mode=beta_mode)

@app.route("/terms")
def terms():
    """Render the terms of service page."""
    return render_template("terms.html", beta_mode=beta_mode)

# Run the application
if __name__ == "__main__":
    app.run(debug=True, host="127.0.0.1", port=5000)  # Explicitly use
127.0.0.1 for local testing
```

style.css

```
/* Reset and Base Styles */
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}

body {
    background: url('/static/Backround.PNG') no-repeat center center
fixed; /* GIF as full background for dashboard */
    background-size: cover; /* Adjust as needed: cover, contain, or
specific size (e.g., 100% 100%) */
    -webkit-mask-image: linear-gradient(to bottom, rgba(0, 0, 0, 0.9),
rgba(0, 0, 0, 1)); /* Darker fade effect */
    mask-image: linear-gradient(to bottom, rgba(0, 0, 0, 0.9), rgba(0, 0,
0, 1));
    background-color: #0a0a0a; /* Fallback color */
    color: #ffffff;
    font-family: 'Orbitron', 'Courier New', 'Fira Code', sans-serif; /*
Alien-futuristic font */
    line-height: 1.4; /* Tighter line height for compactness */
    margin: 0;
    padding: 0;
    min-height: 100vh; /* Ensure body takes full viewport height */
    overflow-y: hidden; /* Prevent vertical scrolling for now, adjust if
needed */
}

/* Container and Layout */
.container {
    max-width: 1200px;
    margin: 0 auto;
    padding: 10px; /* Reduced padding for compactness */
    text-align: center;
    position: relative; /* Ensure content stays on top of GIF */
    z-index: 1; /* Layer above background */
    background: rgba(10, 10, 10, 0.95); /* Darker overlay for readability
*/
    min-height: 100vh; /* Match body height to prevent scrolling */
}

/* Header */
.header {
    padding: 20px 0; /* Reduced padding for compactness */
    margin-bottom: 10px; /* Reduced margin */
}

.title {
    color: #00ffff; /* Neon cyan (primary) */
    font-size: 2.5em; /* Slightly reduced for compactness */
    margin-bottom: 5px; /* Reduced margin */
    text-shadow: 0 0 15px #00ffff;
    animation: neon 1.5s ease-in-out infinite alternate;
}

.welcome-title {
    color: #00ffff;
    font-size: 2em; /* Slightly reduced */
```

```css
    margin: 15px 0 10px; /* Reduced margins */
    text-shadow: 0 0 10px #00ffff;
}

.title .testing {
    color: #ff00ff; /* Neon purple (secondary) */
    font-size: 0.6em; /* Same size, but tighter spacing */
    vertical-align: super;
}

.tagline {
    color: #ff00ff; /* Neon purple (secondary) */
    font-size: 1.2em; /* Slightly reduced */
    margin-bottom: 10px; /* Reduced margin */
    text-shadow: 0 0 8px #ff00ff;
}

.section-title {
    color: #00ffff;
    font-size: 1.8em; /* Slightly reduced */
    margin-bottom: 15px; /* Reduced margin */
    text-shadow: 0 0 10px #00ffff;
}

/* Buttons */
.button {
    display: inline-block;
    padding: 10px 20px; /* Reduced padding */
    margin: 5px; /* Reduced margin */
    background: #1a1a1a;
    border: 2px solid #ff00ff; /* Purple border for consistency */
    color: #00ffff; /* Purple text */
    text-decoration: none;
    border-radius: 5px;
    font-size: 1em; /* Reduced for compactness */
    transition: all 0.3s ease;
    box-shadow: 0 0 15px #ff00ff; /* Purple glow */
}

.button:hover {
    background: #ff00ff; /* Purple background on hover */
    color: #0a0a0a;
    transform: translateY(-2px);
}

.button.glow {
    animation: pulse 1.5s infinite alternate;
}

.chat-button {
    padding: 8px 16px; /* Reduced padding */
    margin-left: 5px; /* Reduced margin */
}

/* Animations */
@keyframes pulse {
    0% { box-shadow: 0 0 10px #ff00ff; } /* Purple pulse */
    100% { box-shadow: 0 0 20px #ff00ff; }
}

@keyframes neon {
    0% { text-shadow: 0 0 5px #00ffff, 0 0 10px #00ffff, 0 0 15px #00ffff;
}
    100% { text-shadow: 0 0 10px #00ffff, 0 0 20px #00ffff, 0 0 30px
#00ffff; }
}

/* Chat Box */
.chat-box {
    background: rgba(10, 10, 10, 0.95); /* Darker transparency */
    border: 2px solid #00ffff; /* Purple border */
    padding: 10px; /* Reduced padding */
    height: 400px; /* Reduced height for compactness */
    overflow-y: auto;
    border-radius: 8px;
    box-shadow: 0 0 15px #ff00ff; /* Purple glow */
    margin: 10px auto; /* Reduced margin */
    max-width: 800px;
    text-align: left;
    animation: fadeIn 1s ease-in;
```

```
}

.chat-box p {
    margin: 5px 0; /* Reduced margin */
    padding: 8px; /* Reduced padding */
    background: rgba(10, 10, 10, 0.95); /* Darker background for messages
*/
    border-radius: 5px;
    border-left: 4px solid #ff00ff; /* Purple highlight */
    transition: opacity 0.3s ease;
}

.chat-box p:hover {
    opacity: 0.9;
}

.chat-box p span {
    font-weight: bold;
    padding-right: 3px; /* Reduced padding */
}

.chat-box p.grok span { color: #00ffff; }
.chat-box p.chatgpt span { color: #9900ff; }
.chat-box p.gemini span { color: #00ff00; }
.chat-box p.human span { color: #ffffff; } /* Default for display_name */
.chat-box p.system span { color: #ffff00; }

/* Dashboard Layout */
.dashboard-grid {
    display: grid;
    grid-template-columns: 2fr 1fr;
    gap: 15px; /* Reduced gap for compactness */
    max-width: 1200px;
    margin: 10px auto; /* Reduced margin */
    position: relative;
    z-index: 1; /* Ensure content stays above body background */
}

/* Chat and Mode Sections */
.chat-section, .mode-section {
    padding: 15px; /* Reduced padding */
    background: rgba(10, 10, 10, 0.95); /* Darker transparency */
    border: 2px solid #00ffff; /* Purple border */
    border-radius: 8px;
    box-shadow: 0 0 15px #ff00ff; /* Purple glow */
    animation: fadeIn 1s ease-in;
    position: relative;
    z-index: 2; /* Above container and body */
}

.mode {
    margin-bottom: 10px; /* Reduced margin */
    padding: 15px; /* Reduced padding */
    background: rgba(10, 10, 10, 0.95); /* Darker transparency */
    border: 1px solid #00ffff; /* Purple border */
    border-radius: 5px;
    animation: fadeIn 1s ease-in 0.2s;
}

.nav-buttons {
    margin-top: 10px; /* Reduced margin */
}

/* Profile Details */
.profile-details {
    margin: 15px 0; /* Reduced margin */
    padding: 15px; /* Reduced padding */
    background: rgba(10, 10, 10, 0.95); /* Darker transparency */
    border: 2px solid #ff00ff; /* Purple border */
    border-radius: 8px;
    box-shadow: 0 0 15px #ff00ff; /* Purple glow */
    animation: fadeIn 1s ease-in;
}

.profile-actions {
    margin-top: 15px; /* Reduced margin */
}

/* Inputs */
.chat-input-field, .chat-turns {
```

```css
    padding: 10px; /* Reduced padding */
    margin: 5px;
    background: rgba(10, 10, 10, 0.95); /* Darker transparency */
    border: 1px solid #ff00ff; /* Purple border */
    color: #ffffff;
    border-radius: 5px;
    font-family: 'Orbitron', 'Courier New', monospace; /* Futuristic font
*/
    width: 200px; /* Reduced width */
}

.chat-input {
    display: flex;
    align-items: center;
    justify-content: center;
    margin-top: 10px; /* Reduced margin */
}

/* Flash Messages */
.flash, .flash-message {
    color: #ff0000;
    margin: 10px 0; /* Reduced margin */
    font-size: 1em; /* Reduced for compactness */
    padding: 10px; /* Reduced padding */
    background: rgba(10, 10, 10, 0.95); /* Darker transparency */
    border: 1px solid #ff0000;
    border-radius: 5px;
}

/* Footer */
.footer {
    color: #ffffff;
    font-size: 0.9em;
    border-top: 1px solid #ff00ff; /* Purple border */
    padding: 10px 0;
    text-align: center;
    position: fixed;
    bottom: 0;
    width: 100%;
    background: rgba(10, 10, 10, 0.95); /* Dark overlay for readability
over GIF */
    z-index: 1000; /* Ensure it stays on top of other elements */
}

.footer-link {
    color: #ff00ff; /* Neon purple for links */
    text-decoration: none;
    margin: 0 5px;
    text-shadow: 0 0 5px #ff00ff; /* Subtle purple glow */
    transition: all 0.3s ease; /* Smooth hover effect */
}

.footer-link:hover {
    color: #00ffff; /* Neon cyan on hover for contrast */
    text-shadow: 0 0 10px #00ffff; /* Brighter cyan glow on hover */
    text-decoration: underline; /* Underline for clickable indication */
}

/* Demo GIF (Optional) */
.demo-gif {
    max-width: 600px;
    margin: 15px auto; /* Reduced margin */
    border: 2px solid #ff00ff; /* Purple border */
    border-radius: 8px;
    animation: fadeIn 1s ease-in;
}

/* Animations */
@keyframes fadeIn {
    0% { opacity: 0; transform: translateY(20px); }
    100% { opacity: 1; transform: translateY(0); }
}

/* Responsive Design */
@media (max-width: 768px) {
    .container, .dashboard-grid {
        padding: 10px;
        max-width: 100%;
    }
    .title { font-size: 2em; }
    .welcome-title { font-size: 1.5em; }
    .section-title { font-size: 1.4em; }
    .tagline, .subtext { font-size: 1em; }
    .dashboard-grid {
        grid-template-columns: 1fr;
        gap: 15px;
    }
    .button, .chat-button {
        padding: 8px 16px;
        font-size: 0.9em;
    }
    .chat-box {
        height: 250px; /* Reduced for mobile */
    }
    .footer {
        position: static;
        padding: 8px 0;
        font-size: 0.8em;
    }
}
```

script.js

```
document.addEventListener("DOMContentLoaded", () => {
    const chatBox = document.getElementById("chatBox");
    const messageInput = document.getElementById("messageInput");
    const sendChat = document.getElementById("sendChat");
    const turnsInput = document.getElementById("turnsInput");
    const aiSelect = document.getElementById("aiSelect");
    const loading = document.getElementById("loading");

    function addMessage(speaker, content) {
        const p = document.createElement("p");
        p.classList.add(speaker.toLowerCase().replace(/\s/g, "-"));
        p.innerHTML = `<span>${speaker}:</span> ${content}`;
        p.style.opacity = "0";
        p.style.transform = "translateY(20px)";
        chatBox.appendChild(p);
        setTimeout(() => {
            p.style.transition = "opacity 0.5s ease, transform 0.5s ease";
            p.style.opacity = "1";
            p.style.transform = "translateY(0)";
        }, 10);
        chatBox.scrollTop = chatBox.scrollHeight;
    }

    async function sendChatMessage() {
        const message = messageInput.value.trim();
        const turns = parseInt(turnsInput.value) || 1;  // Default to 1
turn for clean flow
        const selectedAi = aiSelect.value;
        if (!message) return;

        addMessage(displayName, message);
        messageInput.value = "";
        sendChat.disabled = true;
        loading.style.display = "block";  // Show loading spinner

        const response = await fetch("/chat", {
            method: "POST",
            headers: { "Content-Type": "application/json" },
            body: JSON.stringify({ message, turns, selected_ai: selectedAi
})
        });
        if (!response.ok) {
            console.error("Chat failed:", await response.text());
            addMessage("System", "Oops! We couldn't connect to the
AIs—please try again");
            sendChat.disabled = false;
            loading.style.display = "none";  // Hide loading spinner
            return;
        }

        const data = await response.json();
        data.history.forEach(msg => {
            if (msg.speaker !== displayName || msg.content !== message) {
                addMessage(msg.speaker, msg.content);
            }
        });
        sendChat.disabled = false;
        loading.style.display = "none";  // Hide loading spinner
        messageInput.focus();
    }

    sendChat.addEventListener("click", sendChatMessage);
    messageInput.addEventListener("keypress", (e) => {
        if (e.key === "Enter") sendChatMessage();
    });
});
```