**Explanation of the Flask Application Code**

This Flask application provides a simple authentication flow using Google OAuth and JWT for session management. Here’s an overview of the key components and their functionalities:

* **Setup and Configuration:**
  + The Flask application is configured with the necessary secret key and OAuth settings to integrate with Google.
  + The OAuth client is registered with Google’s OAuth 2.0 API using client credentials, including the client ID, client secret, and other necessary URLs.
* **Index Route (/):**
  + This is the main landing page of the application.
  + It includes a "Sign in with Google" button styled to resemble the Google sign-in button.
  + The **checkLogin** JavaScript function checks if a JWT is stored in local storage. If a JWT is found, it sends a request to the **/get\_user\_info** endpoint to verify the token and fetch user information. If the JWT is valid, the user is redirected to the **/welcome** page.
* **Login Route (/login):**
  + This route initiates the Google OAuth 2.0 authorization flow.
  + A nonce (number used once) is generated and stored in the session to prevent replay attacks.
  + The user is redirected to Google’s authorization URL.
* **Callback Route (/login/callback):**
  + This route handles the callback from Google after the user has authenticated.
  + The ID token received from Google is sent to a Lambda function to authenticate the user and generate a JWT.
  + The JWT received from the Lambda function is stored in the browser’s local storage, and the user is redirected to the **/welcome** page.
* **Welcome Route (/welcome):**
  + This route displays a welcome message along with the user’s name and email.
  + The **getUserInfo** JavaScript function fetches user information from the **/get\_user\_info** endpoint using the JWT stored in local storage. If the JWT is valid, the user’s name and email are displayed. Otherwise, the user is logged out.
* **Get User Info Route (/get\_user\_info):**
  + This endpoint receives a JWT and sends it to a Lambda function to fetch user information.
  + If the JWT is valid, the user information is returned; otherwise, an error response is returned.
* **JavaScript Functions:**
  + **loginWithGoogle()**: Redirects the user to the **/login** route to initiate the Google login flow.
  + **checkLogin()**: Checks if a JWT is stored in local storage and verifies it by calling the **/get\_user\_info** endpoint.
  + **getUserInfo()**: Fetches user information using the stored JWT and updates the welcome page.
* **API Endpoints:**
  + <https://49vrl95j02.execute-api.us-east-1.amazonaws.com/prod1/users>
  + headers = {'Content-Type': 'application/json'}
  + response = requests.post(lambda\_url, json={'idToken':?? headers=???)

**Key Points:**

* **OAuth Integration:** Uses Google’s OAuth 2.0 for user authentication.
* **JWT Management:** Manages user sessions using JWTs stored in the browser’s local storage.
* **Serverless Integration:** Interacts with a Lambda function to validate tokens and fetch user information.
* **User Experience:** Provides a smooth user experience with styled buttons and seamless redirection based on authentication status.

This setup ensures secure authentication and a streamlined user experience, leveraging modern authentication standards and serverless technologies.