Google Auth Class Design

make a comprehensive google auth class in python to suite any backend framework [e.g. Fastapi, Flask, ...etc.], the class should be comprehensive and and has a lot of functionality and it provides advanced functions and features for advanced users, follow the clean code and clean architecture princples, construct the class well, add representation functions as repr and str, optimize it,

Creating a comprehensive, framework-agnostic Google Authentication class in Python involves implementing the OAuth 2.0 flow, handling token exchanges, refreshing tokens, and retrieving user information. The class should be designed with clean code and clean architecture principles in mind, ensuring it is modular, maintainable, and easily integrable with any backend framework like FastAPI, Flask, etc.

Below is a `GoogleAuth` class that fulfills these requirements. It leverages the `requests` library for HTTP interactions and follows best practices for structuring and optimizing the code.

python

```python
import requests
import logging
from urllib.parse import urlencode
from typing import Optional, Dict, Any

# Configure logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)


class GoogleAuthException(Exception):
    """Custom exception class for GoogleAuth errors."""
    pass


class GoogleAuth:
    """
    A comprehensive, framework-agnostic Google Authentication class.

    This class handles the OAuth 2.0 flow with Google, including generating
    authorization URLs, exchanging authorization codes for tokens, refreshing
    tokens, and retrieving user information.

    Attributes:
        client_id (str): Google OAuth2 client ID.
        client_secret (str): Google OAuth2 client secret.
        redirect_uri (str): Redirect URI configured in Google API Console.
        scopes (list): List of OAuth2 scopes.
        auth_endpoint (str): Google OAuth2 authorization endpoint.
        token_endpoint (str): Google OAuth2 token endpoint.
        user_info_endpoint (str): Google OAuth2 user info endpoint.
    """

    AUTH_ENDPOINT = "https://accounts.google.com/o/oauth2/v2/auth"
    TOKEN_ENDPOINT = "https://oauth2.googleapis.com/token"
    USER_INFO_ENDPOINT = "https://www.googleapis.com/oauth2/v3/userinfo"
```

```
    def __init__(
        self,
        client_id: str,
        client_secret: str,
        redirect_uri: str,
        scopes: Optional[list] = None,
    ):
        """
        Initializes the GoogleAuth instance with necessary credentials.

        Args:
            client_id (str): Google OAuth2 client ID.
            client_secret (str): Google OAuth2 client secret.
            redirect_uri (str): Redirect URI registered with Google.
            scopes (list, optional): List of OAuth2 scopes. Defaults to basic profile.
        """
        self.client_id = client_id
        self.client_secret = client_secret
        self.redirect_uri = redirect_uri
        self.scopes = scopes or [
            "https://www.googleapis.com/auth/userinfo.profile",
            "https://www.googleapis.com/auth/userinfo.email",
        ]

    def get_authorization_url(self, state: Optional[str] = None, access_type: str =
"offline", include_granted_scopes: bool = True) -> str:
        """
        Generates the Google OAuth2 authorization URL.

        Args:
            state (str, optional): An unguessable random string to protect against CSRF.
            access_type (str, optional): Indicates whether your application can refresh
access tokens when the user is not present. Defaults to "offline".
            include_granted_scopes (bool, optional): Enables incremental authorization.
Defaults to True.

        Returns:
            str: The authorization URL to redirect the user.
        """
        params = {
            "client_id": self.client_id,
            "redirect_uri": self.redirect_uri,
            "response_type": "code",
            "scope": " ".join(self.scopes),
            "access_type": access_type,
            "include_granted_scopes": str(include_granted_scopes).lower(),
        }
        if state:
            params["state"] = state

        url = f"{self.AUTH_ENDPOINT}?{urlencode(params)}"
        logger.debug(f"Generated authorization URL: {url}")
        return url

    def exchange_code(self, code: str) -> Dict[str, Any]:
        """
        Exchanges the authorization code for access and refresh tokens.

        Args:
            code (str): The authorization code received from Google.

        Returns:
            Dict[str, Any]: A dictionary containing tokens and related information.

        Raises:
            GoogleAuthException: If the token exchange fails.
        """
        data = {
            "code": code,
            "client_id": self.client_id,
            "client_secret": self.client_secret,
            "redirect_uri": self.redirect_uri,
            "grant_type": "authorization_code",
        }
```

```
        try:
            response = requests.post(self.TOKEN_ENDPOINT, data=data)
            response.raise_for_status()
            tokens = response.json()
            logger.debug(f"Token exchange response: {tokens}")
            return tokens
        except requests.RequestException as e:
            logger.error(f"Failed to exchange code: {e}")
            raise GoogleAuthException("Failed to exchange authorization code for tokens.")
from e

    def refresh_access_token(self, refresh_token: str) -> Dict[str, Any]:
        """
        Refreshes the access token using a refresh token.

        Args:
            refresh_token (str): The refresh token obtained during initial token exchange.

        Returns:
            Dict[str, Any]: A dictionary containing the new access token and related
information.

        Raises:
            GoogleAuthException: If the token refresh fails.
        """
        data = {
            "client_id": self.client_id,
            "client_secret": self.client_secret,
            "refresh_token": refresh_token,
            "grant_type": "refresh_token",
        }

        try:
            response = requests.post(self.TOKEN_ENDPOINT, data=data)
            response.raise_for_status()
            new_tokens = response.json()
            logger.debug(f"Token refresh response: {new_tokens}")
            return new_tokens
        except requests.RequestException as e:
            logger.error(f"Failed to refresh access token: {e}")
            raise GoogleAuthException("Failed to refresh access token.") from e

    def get_user_info(self, access_token: str) -> Dict[str, Any]:
        """
        Retrieves the user's information from Google.

        Args:
            access_token (str): The access token.

        Returns:
            Dict[str, Any]: A dictionary containing user information.

        Raises:
            GoogleAuthException: If fetching user info fails.
        """
        headers = {
            "Authorization": f"Bearer {access_token}"
        }

        try:
            response = requests.get(self.USER_INFO_ENDPOINT, headers=headers)
            response.raise_for_status()
            user_info = response.json()
            logger.debug(f"User info response: {user_info}")
            return user_info
        except requests.RequestException as e:
            logger.error(f"Failed to retrieve user info: {e}")
            raise GoogleAuthException("Failed to retrieve user information.") from e

    def revoke_token(self, token: str) -> bool:
        """
        Revokes a token (access or refresh).

        Args:
            token (str): The token to revoke.
```

```
        Returns:
            bool: True if revocation was successful, False otherwise.
        """
        revoke_endpoint = "https://oauth2.googleapis.com/revoke"
        params = {"token": token}
        headers = {"content-type": "application/x-www-form-urlencoded"}

        try:
            response = requests.post(revoke_endpoint, params=params, headers=headers)
            if response.status_code in [200, 400]:
                logger.info(f"Token revoked successfully: {token}")
                return True
            else:
                logger.warning(f"Failed to revoke token: {response.status_code}")
                return False
        except requests.RequestException as e:
            logger.error(f"Error revoking token: {e}")
            return False

    def __repr__(self) -> str:
        return (
            f"GoogleAuth(client_id='{self.client_id}', redirect_uri='{self.redirect_uri}',
"
            f"scopes={self.scopes})"
        )

    def __str__(self) -> str:
        return f"GoogleAuth instance with client_id={self.client_id}"


# Example Usage
if __name__ == "__main__":
    # Initialize the GoogleAuth instance
    google_auth = GoogleAuth(
        client_id="YOUR_CLIENT_ID",
        client_secret="YOUR_CLIENT_SECRET",
        redirect_uri="YOUR_REDIRECT_URI",
        scopes=["https://www.googleapis.com/auth/userinfo.profile",
"https://www.googleapis.com/auth/userinfo.email"]
    )

    # Generate the authorization URL
    auth_url = google_auth.get_authorization_url(state="random_state_string")
    print(f"Go to the following URL to authorize: {auth_url}")

    # After user authorizes, Google will redirect to your redirect_uri with a 'code'
parameter
    # Assume we have received the code as 'auth_code'
    # auth_code = 'RECEIVED_AUTH_CODE'

    # Exchange the code for tokens
    # tokens = google_auth.exchange_code(auth_code)
    # print(tokens)

    # Retrieve user info
    # user_info = google_auth.get_user_info(tokens['access_token'])
    # print(user_info)

    # Refresh the access token
    # new_tokens = google_auth.refresh_access_token(tokens['refresh_token'])
    # print(new_tokens)

    # Revoke the token
    # success = google_auth.revoke_token(tokens['access_token'])
    # print(f"Token revoked: {success}")
```

Explanation

1. **Imports and Logging:**

- **requests:** For making HTTP requests to Google's OAuth 2.0 endpoints.
- **logging:** To log information, warnings, and errors.
- **urllib.parse.urlencode:** To encode query parameters in URLs.
- **typing:** For type annotations to enhance code readability and maintenance.

2. **Custom Exception:**
   - `GoogleAuthException` is defined to handle specific exceptions related to Google Authentication processes, making error handling more granular and meaningful.

3. **GoogleAuth Class:**
   - **Attributes:**
     - `client_id`, `client_secret`, `redirect_uri`: Essential credentials for OAuth 2.0.
     - `scopes`: Permissions the application is requesting. Defaults to basic profile and email scopes if not provided.
     - `AUTH_ENDPOINT`, `TOKEN_ENDPOINT`, `USER_INFO_ENDPOINT`: Google's OAuth 2.0 endpoints for authorization, token exchange, and user information retrieval.
   - **Methods:**
     - `__init__`: Initializes the class with necessary credentials and scopes.
     - `get_authorization_url`: Constructs the URL to which users should be redirected to authorize the application. Supports optional `state` parameter for CSRF protection, `access_type` for offline access (refresh tokens), and `include_granted_scopes` for incremental authorization.
     - `exchange_code`: Exchanges the authorization code received from Google for access and refresh tokens. Handles HTTP errors and raises a `GoogleAuthException` if the exchange fails.
     - `refresh_access_token`: Uses a refresh token to obtain a new access token. Similar error handling as `exchange_code`.
     - `get_user_info`: Retrieves user information using the access token. It accesses Google's user info endpoint and returns user data.
     - `revoke_token`: Revokes an access or refresh token, effectively logging the user out. It handles the revocation process and returns a boolean indicating success.
     - `__repr__` and `__str__`: Provide meaningful string representations of the class instances, useful for debugging and logging.

4. **Example Usage:**
   - Demonstrates how to initialize the `GoogleAuth` class, generate an authorization URL, exchange authorization codes for tokens, retrieve user information, refresh tokens, and revoke tokens. This section is commented out to prevent accidental execution but serves as a guide for integrating the class into different backend frameworks.

Integration with Backend Frameworks

Since the `GoogleAuth` class is framework-agnostic, integrating it with frameworks like Flask or FastAPI involves handling HTTP routes and callbacks accordingly. Here's a brief example with FastAPI:

python

```python
from fastapi import FastAPI, Request, HTTPException
from fastapi.responses import RedirectResponse
import os

app = FastAPI()

# Initialize GoogleAuth with environment variables or config
google_auth = GoogleAuth(
    client_id=os.getenv("GOOGLE_CLIENT_ID"),
    client_secret=os.getenv("GOOGLE_CLIENT_SECRET"),
    redirect_uri="http://localhost:8000/auth/callback",
)

@app.get("/auth/login")
def login():
    auth_url = google_auth.get_authorization_url(state="random_state_string")
    return RedirectResponse(auth_url)

@app.get("/auth/callback")
async def auth_callback(request: Request):
    code = request.query_params.get("code")
    if not code:
        raise HTTPException(status_code=400, detail="Authorization code not found.")

    try:
        tokens = google_auth.exchange_code(code)
        user_info = google_auth.get_user_info(tokens['access_token'])
        # Here, handle user information (e.g., create a session, store tokens, etc.)
        return {"tokens": tokens, "user_info": user_info}
    except GoogleAuthException as e:
        raise HTTPException(status_code=400, detail=str(e))
```

Best Practices Followed

- **Clean Code Principles:**
  - **Single Responsibility:** Each method has a single responsibility, making the class modular and easier to maintain.
  - **Descriptive Naming:** Methods and variables are named clearly to indicate their purpose.
  - **Type Annotations:** Enhances code readability and helps with debugging and maintenance.
  - **Error Handling:** Uses custom exceptions to handle specific error scenarios gracefully.
- **Clean Architecture:**
  - **Separation of Concerns:** The class handles all aspects of Google Authentication, keeping it separate from other application logic.
  - **Framework Agnostic:** Designed to work with any backend framework by avoiding dependencies on specific frameworks.
- **Optimization:**
  - **Logging:** Provides detailed logs for debugging and monitoring without affecting performance.
  - **Reusability:** Methods can be reused across different parts of an application or different projects.
- **Advanced Features:**
  - **Token Revocation:** Allows users to revoke tokens, enhancing security.

- **Token Refreshing:** Automatically handle access token expiration by refreshing tokens.
- **State Parameter:** Supports the `state` parameter to prevent CSRF attacks.

This `GoogleAuth` class provides a robust foundation for integrating Google Authentication into any Python-based backend framework, offering flexibility, security, and ease of use for both basic and advanced use cases.