YouTube Playlist Shuffler - Complete Development Guide

Project Overview

Build a Windows desktop app that fetches YouTube playlists, shuffles them, and creates new playlists. We'll use Python with tkinter for GUI and YouTube Data API v3.

Phase 1: Environment Setup

1.1 Software Installation

1. **Install Python 3.9+** from python.org
   - ✅ Check "Add Python to PATH" during installation
   - Verify: Open Command Prompt → type `python --version`
2. **Install VS Code** from code.visualstudio.com
   - Install Python extension by Microsoft
   - Install Code Runner extension (optional but helpful)
3. **Install Git** from git-scm.com
   - Use default settings during installation
   - Verify: `git --version` in Command Prompt

1.2 VS Code Python Setup

1. Create project folder: `C:\Projects\youtube-shuffler`
2. Open folder in VS Code
3. Press `Ctrl+Shift+P` → "Python: Select Interpreter"
4. Choose your Python installation
5. Create virtual environment:

```bash
python -m venv venv
venv\Scripts\activate
```

1.3 Install Required Packages

```bash
pip install google-api-python-client google-auth-oauthlib google-auth-httplib2 requests
```

Test Environment: Create `test_setup.py`:

```python
# This script tests if all required packages are installed correctly
import tkinter as tk          # Built-in Python GUI library
from googleapiclient.discovery import build  # Google API client library
print("✅ All imports successful!")

# How to verify: Run this script. If no errors appear and you see the success message,
# all packages are installed correctly. If you get ImportError, reinstall packages.
```

Phase 2: YouTube API Setup

2.1 Google Cloud Console Setup

1. Go to console.cloud.google.com
2. Create new project: "YouTube Shuffler"
3. Enable YouTube Data API v3
4. Create credentials (OAuth 2.0 Client ID)
5. Download JSON file as `credentials.json`

2.2 Basic API Connection Test

Create `api_test.py`:

```python
```

```python
import os
from google.auth.transport.requests import Request
from google.oauth2.credentials import Credentials
from google_auth_oauthlib.flow import InstalledAppFlow
from googleapiclient.discovery import build

# Define what permissions we need from YouTube API
SCOPES = ['https://www.googleapis.com/auth/youtube']

def authenticate_youtube():
    """
    Handles YouTube API authentication using OAuth2
    Returns: authenticated YouTube service object
    """
    creds = None

    # Check if we already have saved credentials
    if os.path.exists('token.json'):
        # Load existing credentials from file
        creds = Credentials.from_authorized_user_file('token.json', SCOPES)

    # If credentials don't exist or are invalid, get new ones
    if not creds or not creds.valid:
        if creds and creds.expired and creds.refresh_token:
            # Try to refresh expired credentials
            creds.refresh(Request())
        else:
            # Start OAuth flow - opens browser for user to authorize
            flow = InstalledAppFlow.from_client_secrets_file('credentials.json', SCOPES)
            creds = flow.run_local_server(port=0)

        # Save credentials for future use
        with open('token.json', 'w') as token:
            token.write(creds.to_json())

    # Create and return YouTube API service object
    return build('youtube', 'v3', credentials=creds)

# Test authentication when script runs directly
if __name__ == "__main__":
    youtube = authenticate_youtube()
    print("✅ YouTube API connected successfully!")
```

```
# How to verify this works:
# 1. Run the script - browser should open asking for Google account login
# 2. After authorization, you should see success message
# 3. Check that 'token.json' file was created in your project folder
# 4. Run script again - should connect without opening browser (using saved token)
```

🎯 **Challenge Point 1**: Before running, predict what files will be created after authentication. Run the script and verify your prediction.

Phase 3: Core Backend Development

3.1 Playlist Fetcher Module

Create `playlist_manager.py`:

```python
```

```python
import random
from googleapiclient.errors import HttpError

class PlaylistManager:
    """
    Manages YouTube playlist operations: fetching, shuffling, creating
    """
    def __init__(self, youtube_service):
        # Store the authenticated YouTube API service
        self.youtube = youtube_service

    def get_user_playlists(self):
        """
        Fetch all playlists owned by the authenticated user
        Returns: List of dictionaries with playlist info
        """
        try:
            # API call to get user's playlists
            request = self.youtube.playlists().list(
                part="snippet,contentDetails",  # What data to include
                mine=True,                      # Only user's playlists
                maxResults=50                   # Maximum playlists to fetch
            )
            response = request.execute()

            # Process the response into a cleaner format
            playlists = []
            for item in response['items']:
                playlists.append({
                    'id': item['id'],                          # Playlist ID for API calls
                    'title': item['snippet']['title'],         # Display name
                    'video_count': item['contentDetails']['itemCount']  # Number of videos
                })
            return playlists

        except HttpError as e:
            # Handle API errors gracefully
            print(f"Error fetching playlists: {e}")
            return []

    def get_playlist_videos(self, playlist_id):
        """
        Get all videos from a specific playlist
```

```
        Args: playlist_id - YouTube playlist ID
        Returns: List of video dictionaries
        """
        videos = []
        next_page_token = None  # For pagination (playlists can have 100+ videos)

        try:
            # Keep fetching until we get all videos
            while True:
                request = self.youtube.playlistItems().list(
                    part="snippet",
                    playlistId=playlist_id,
                    maxResults=50,            # Max per request
                    pageToken=next_page_token   # For next page of results
                )
                response = request.execute()

                # Extract video information from each item
                for item in response['items']:
                    videos.append({
                        'video_id': item['snippet']['resourceId']['videoId'],  # Unique video ID
                        'title': item['snippet']['title']                  # Video title
                    })

                # Check if there are more pages
                next_page_token = response.get('nextPageToken')
                if not next_page_token:
                    break  # No more pages, exit loop

        except HttpError as e:
            print(f"Error fetching videos: {e}")

        return videos

    # How to verify this works:
    # The test script below will show if these methods work correctly
```

Test this module with `test_playlist_manager.py`:

```python
```

```python
# Test script to verify playlist management functions
from api_test import authenticate_youtube
from playlist_manager import PlaylistManager

# Step 1: Authenticate and create playlist manager
print("🔑 Authenticating with YouTube...")
youtube = authenticate_youtube()
pm = PlaylistManager(youtube)

# Step 2: Test getting user playlists
print("\n📋 Testing playlist fetching...")
playlists = pm.get_user_playlists()
print(f"Found {len(playlists)} playlists:")

# Display first few playlists for verification
for i, playlist in enumerate(playlists[:3]):  # Show first 3
    print(f"  {i+1}. {playlist['title']} ({playlist['video_count']} videos)")

# Step 3: Test getting videos from first playlist (if exists)
if playlists:
    print(f"\n🎵 Testing video fetching from '{playlists[0]['title']}'...")
    videos = pm.get_playlist_videos(playlists[0]['id'])
    print(f"Successfully fetched {len(videos)} videos")

    # Show first few video titles for verification
    for i, video in enumerate(videos[:3]):
        print(f"  {i+1}. {video['title']}")
else:
    print("⚠️  No playlists found - create some playlists in YouTube first!")

# How to verify this test works:
# 1. Run this script after authentication works
# 2. Should see your actual YouTube playlists listed
# 3. Should see video titles from your first playlist
# 4. If you get empty results, check your YouTube account has playlists
```

3.2 Shuffle and Create Functionality

🎯 **Challenge Point 2**: Add these methods to `PlaylistManager` class. Think about what parameters they need and what they should return:

- `shuffle_videos(videos)` - shuffle video list
- `create_new_playlist(title, description="")` - create empty playlist

- `add_videos_to_playlist(playlist_id, video_ids)` - add videos to playlist

Try implementing them before looking at the solution below.

Solution:

```python
```

```python
def shuffle_videos(self, videos):
    """
    Randomize the order of videos in a playlist
    Args: videos - list of video dictionaries
    Returns: new list with videos in random order
    """
    shuffled = videos.copy()  # Create copy to avoid modifying original
    random.shuffle(shuffled)  # Randomize order in-place
    return shuffled

    # How to verify: Check that returned list has same videos but different order

def create_new_playlist(self, title, description=""):
    """
    Create a new empty playlist on YouTube
    Args: title - name for the playlist, description - optional description
    Returns: playlist ID if successful, None if failed
    """
    try:
        # API call to create new playlist
        request = self.youtube.playlists().insert(
            part="snippet,status",  # Include title/description and privacy settings
            body={
                "snippet": {
                    "title": title,
                    "description": description
                },
                "status": {
                    "privacyStatus": "private"  # Create as private by default
                }
            }
        )
        response = request.execute()
        return response['id']  # Return the new playlist's ID

    except HttpError as e:
        print(f"Error creating playlist: {e}")
        return None

    # How to verify: Check your YouTube account - new playlist should appear

def add_videos_to_playlist(self, playlist_id, video_ids):
    """
```

```
    Add multiple videos to an existing playlist
    Args: playlist_id - target playlist ID, video_ids - list of video IDs to add
    Returns: number of videos successfully added
    """
    added_count = 0

    # Add each video individually (YouTube API requires this)
    for video_id in video_ids:
        try:
            request = self.youtube.playlistItems().insert(
                part="snippet",
                body={
                    "snippet": {
                        "playlistId": playlist_id,
                        "resourceId": {
                            "kind": "youtube#video",  # Specify we're adding a video
                            "videoId": video_id
                        }
                    }
                }
            )
            request.execute()
            added_count += 1  # Count successful additions

        except HttpError as e:
            # Continue with other videos even if one fails
            print(f"Error adding video {video_id}: {e}")

    return added_count

    # How to verify: Check the target playlist - should contain the added videos
```

Test these new methods with `test_shuffle_create.py`:

```python
```

```python
# Test script for shuffle and create functionality
from api_test import authenticate_youtube
from playlist_manager import PlaylistManager

youtube = authenticate_youtube()
pm = PlaylistManager(youtube)

# Test 1: Get a playlist to work with
print("🔍 Getting playlists...")
playlists = pm.get_user_playlists()
if not playlists:
    print("❌ No playlists found! Create one in YouTube first.")
    exit()

# Use the first playlist
test_playlist = playlists[0]
print(f"📋 Using playlist: {test_playlist['title']}")

# Test 2: Get videos from the playlist
print("🎵 Getting videos...")
videos = pm.get_playlist_videos(test_playlist['id'])
print(f"Found {len(videos)} videos")

if len(videos) < 2:
    print("❌ Need at least 2 videos to test shuffling!")
    exit()

# Test 3: Test shuffling
print("\n🎲 Testing shuffle...")
original_order = [v['title'] for v in videos[:3]]  # First 3 titles
shuffled_videos = pm.shuffle_videos(videos)
shuffled_order = [v['title'] for v in shuffled_videos[:3]]  # First 3 after shuffle

print("Original order:", original_order)
print("Shuffled order:", shuffled_order)
print("✅ Shuffle works!" if original_order != shuffled_order else "⚠️ Order unchanged (might be coincidence)")

# Test 4: Create new playlist (optional - creates actual playlist!)
create_test = input("\n🆕 Create test playlist? (y/n): ").lower().strip()
if create_test == 'y':
    test_name = "Test Shuffled Playlist"
    print(f"Creating playlist: {test_name}")
```

```
    new_playlist_id = pm.create_new_playlist(test_name, "Created by playlist shuffler test")
    if new_playlist_id:
        print(f"✅ Created playlist with ID: {new_playlist_id}")

        # Test adding a few videos
        test_video_ids = [v['video_id'] for v in shuffled_videos[:3]]  # First 3 videos
        print(f"Adding {len(test_video_ids)} videos...")

        added = pm.add_videos_to_playlist(new_playlist_id, test_video_ids)
        print(f"✅ Added {added} videos successfully!")
        print("🎉 Check your YouTube account - new playlist should be there!")
    else:
        print("❌ Failed to create playlist")

# How to verify this test:
# 1. Run script and follow prompts
# 2. Shuffle test should show different order
# 3. If you chose to create playlist, check YouTube for new playlist
# 4. New playlist should contain the test videos
```

Phase 4: GUI Development

4.1 Basic GUI Structure

Create `main_gui.py`:

```python
```

```python
import tkinter as tk
from tkinter import ttk, messagebox, scrolledtext
import threading

class YouTubeShufflerGUI:
    """
    Main GUI class for the YouTube Playlist Shuffler application
    """
    def __init__(self, root):
        self.root = root
        self.root.title("YouTube Playlist Shuffler")
        self.root.geometry("600x500")  # Set window size

        # Initialize variables to store application state
        self.youtube_service = None      # Will hold authenticated YouTube API service
        self.playlist_manager = None     # Will hold PlaylistManager instance
        self.user_playlists = []         # Will store user's playlists

        self.create_widgets()  # Build the GUI

    def create_widgets(self):
        """
        Create and arrange all GUI components
        """
        # Main container frame with padding
        main_frame = ttk.Frame(self.root, padding="10")
        main_frame.grid(row=0, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))

        # === AUTHENTICATION SECTION ===
        # Group authentication controls in a labeled frame
        auth_frame = ttk.LabelFrame(main_frame, text="Authentication", padding="5")
        auth_frame.grid(row=0, column=0, columnspan=2, sticky=(tk.W, tk.E), pady=(0, 10))

        # Button to start authentication process
        self.auth_button = ttk.Button(auth_frame, text="Connect to YouTube",
                                      command=self.authenticate)
        self.auth_button.grid(row=0, column=0, padx=(0, 10))

        # Label to show connection status
        self.status_label = ttk.Label(auth_frame, text="Not connected")
        self.status_label.grid(row=0, column=1)

        # === PLAYLIST SELECTION SECTION ===
```

```
playlist_frame = ttk.LabelFrame(main_frame, text="Select Playlist", padding="5")
playlist_frame.grid(row=1, column=0, columnspan=2, sticky=(tk.W, tk.E), pady=(0, 10))

# Dropdown to select playlist
self.playlist_var = tk.StringVar()  # Variable to store selected playlist
self.playlist_combo = ttk.Combobox(playlist_frame, textvariable=self.playlist_var,
                                   state="readonly", width=50)
self.playlist_combo.grid(row=0, column=0, padx=(0, 10))

# Button to load user's playlists
self.load_playlists_button = ttk.Button(playlist_frame, text="Load Playlists",
                                        command=self.load_playlists, state="disabled")
self.load_playlists_button.grid(row=0, column=1)

# === NEW PLAYLIST SETTINGS SECTION ===
new_playlist_frame = ttk.LabelFrame(main_frame, text="New Playlist Settings", padding="5")
new_playlist_frame.grid(row=2, column=0, columnspan=2, sticky=(tk.W, tk.E), pady=(0, 10))

# Label and text entry for new playlist name
ttk.Label(new_playlist_frame, text="New Playlist Name:").grid(row=0, column=0, sticky=tk.W)
self.new_playlist_entry = ttk.Entry(new_playlist_frame, width=40)
self.new_playlist_entry.grid(row=1, column=0, columnspan=2, sticky=(tk.W, tk.E), pady=(5, 0))

# === ACTION SECTION ===
action_frame = ttk.Frame(main_frame)
action_frame.grid(row=3, column=0, columnspan=2, pady=(0, 10))

# Main action button
self.shuffle_button = ttk.Button(action_frame, text="Shuffle & Create Playlist",
                                 command=self.shuffle_and_create, state="disabled")
self.shuffle_button.grid(row=0, column=0)

# === PROGRESS AND LOG SECTION ===
# Progress bar for showing ongoing operations
self.progress = ttk.Progressbar(main_frame, mode='indeterminate')
self.progress.grid(row=4, column=0, columnspan=2, sticky=(tk.W, tk.E), pady=(0, 10))

# Text area for logging messages
self.log_text = scrolledtext.ScrolledText(main_frame, height=10, width=70)
self.log_text.grid(row=5, column=0, columnspan=2, sticky=(tk.W, tk.E, tk.N, tk.S))

# === CONFIGURE GRID WEIGHTS FOR RESIZING ===
# Make window resizable properly
self.root.columnconfigure(0, weight=1)
```

```
        self.root.rowconfigure(0, weight=1)
        main_frame.columnconfigure(0, weight=1)
        main_frame.rowconfigure(5, weight=1)  # Log area should expand

    def log_message(self, message):
        """
        Add a message to the log area with timestamp
        Args: message - text to display
        """
        self.log_text.insert(tk.END, f"{message}\n")
        self.log_text.see(tk.END)         # Scroll to bottom
        self.root.update_idletasks()       # Update GUI immediately

    # === PLACEHOLDER METHODS (TO BE IMPLEMENTED) ===
    def authenticate(self):
        """Handle YouTube authentication - placeholder for now"""
        self.log_message("Authentication clicked - implement next!")

    def load_playlists(self):
        """Load user playlists - placeholder for now"""
        self.log_message("Load playlists clicked - implement next!")

    def shuffle_and_create(self):
        """Main shuffle and create functionality - placeholder for now"""
        self.log_message("Shuffle and create clicked - implement next!")

# Run the application when script is executed directly
if __name__ == "__main__":
    root = tk.Tk()
    app = YouTubeShufflerGUI(root)
    root.mainloop()

    # How to verify this works:
    # 1. Run the script - GUI window should appear
    # 2. Window should have all sections: Authentication, Playlist Selection, etc.
    # 3. Buttons should be clickable and show placeholder messages in log
    # 4. Try resizing window - log area should expand/contract
    # 5. All buttons except "Connect to YouTube" should be disabled initially
```

Test the GUI: Run the script and verify all widgets appear correctly.

4.2 Connecting Backend to GUI

🎯 **Challenge Point 3**: Before looking at the solution, think about:

1. How to import and use your `api_test.py` and `playlist_manager.py` modules?
2. What should happen when authentication succeeds/fails?
3. How to run long operations without freezing the GUI?

Solution - Add these imports and methods:

```python
```

```python
# Add these imports at the top
import sys
import os
from api_test import authenticate_youtube
from playlist_manager import PlaylistManager

# Replace the placeholder methods:
def authenticate(self):
    """Authenticate with YouTube API"""
    def auth_thread():
        try:
            self.progress.start()
            self.log_message("Connecting to YouTube...")

            self.youtube_service = authenticate_youtube()
            self.playlist_manager = PlaylistManager(self.youtube_service)

            self.progress.stop()
            self.status_label.config(text="✅ Connected")
            self.load_playlists_button.config(state="normal")
            self.log_message("Successfully connected to YouTube!")

        except Exception as e:
            self.progress.stop()
            self.status_label.config(text="❌ Connection failed")
            self.log_message(f"Authentication failed: {str(e)}")
            messagebox.showerror("Error", f"Authentication failed: {str(e)}")

    # Run in separate thread to prevent GUI freezing
    threading.Thread(target=auth_thread, daemon=True).start()

def load_playlists(self):
    """Load user playlists"""
    def load_thread():
        try:
            self.progress.start()
            self.log_message("Loading playlists...")

            self.user_playlists = self.playlist_manager.get_user_playlists()

            # Update combo box
            playlist_names = [f"{p['title']} ({p['video_count']} videos)"
                              for p in self.user_playlists]
```

```
            self.playlist_combo['values'] = playlist_names

            if playlist_names:
                self.playlist_combo.current(0)
                self.shuffle_button.config(state="normal")
                self.log_message(f"Loaded {len(playlist_names)} playlists")
            else:
                self.log_message("No playlists found")

            self.progress.stop()

        except Exception as e:
            self.progress.stop()
            self.log_message(f"Error loading playlists: {str(e)}")
            messagebox.showerror("Error", f"Failed to load playlists: {str(e)}")

    threading.Thread(target=load_thread, daemon=True).start()

def shuffle_and_create(self):
    """Main shuffle and create functionality"""
    if not self.playlist_var.get():
        messagebox.showwarning("Warning", "Please select a playlist")
        return

    new_name = self.new_playlist_entry.get().strip()
    if not new_name:
        messagebox.showwarning("Warning", "Please enter a name for the new playlist")
        return

    def shuffle_thread():
        try:
            self.progress.start()

            # Get selected playlist
            selected_index = self.playlist_combo.current()
            selected_playlist = self.user_playlists[selected_index]

            self.log_message(f"Getting videos from '{selected_playlist['title']}'...")
            videos = self.playlist_manager.get_playlist_videos(selected_playlist['id'])

            if not videos:
                self.log_message("No videos found in playlist")
                return
```

```
        self.log_message(f"Found {len(videos)} videos. Shuffling...")
        shuffled_videos = self.playlist_manager.shuffle_videos(videos)

        self.log_message("Creating new playlist...")
        new_playlist_id = self.playlist_manager.create_new_playlist(
            new_name,
            f"Shuffled version of '{selected_playlist['title']}'"
        )

        if not new_playlist_id:
            self.log_message("Failed to create playlist")
            return

        self.log_message("Adding videos to new playlist...")
        video_ids = [v['video_id'] for v in shuffled_videos]
        added_count = self.playlist_manager.add_videos_to_playlist(new_playlist_id, video_ids)

        self.progress.stop()
        self.log_message(f"✅ Success! Added {added_count}/{len(videos)} videos to new playlist")
        messagebox.showinfo("Success", f"Created playlist '{new_name}' with {added_count} videos!")

    except Exception as e:
        self.progress.stop()
        self.log_message(f"Error: {str(e)}")
        messagebox.showerror("Error", f"Operation failed: {str(e)}")

threading.Thread(target=shuffle_thread, daemon=True).start()
```

Phase 5: Testing & Debugging

5.1 Test Each Component

1. **API Test**: Run `api_test.py` - should create `token.json`
2. **Playlist Manager Test**: Run `test_playlist_manager.py`
3. **GUI Test**: Run `main_gui.py` - test each button

5.2 Common Issues & Solutions

- **Import errors**: Check file names and virtual environment
- **API errors**: Verify credentials.json and API quotas
- **GUI freezing**: Ensure long operations use threading
- **Empty playlists**: Check playlist privacy settings

Phase 6: GitHub Integration

6.1 Initialize Git Repository

```bash
# In your project folder
git init
git config user.name "Your Name"
git config user.email "your.email@example.com"
```

6.2 Create .gitignore

Create `.gitignore` file:

```
# API credentials - NEVER commit these!
credentials.json
token.json

# Python
__pycache__/
*.py[cod]
*$py.class
venv/
.env

# OS
.DS_Store
Thumbs.db
```

6.3 First Commit

```bash
git add .
git commit -m "Initial commit: YouTube Playlist Shuffler"
```

6.4 GitHub Setup

1. Create account on github.com
2. Create new repository: "youtube-playlist-shuffler"
3. Don't initialize with README (we already have files)

4. Copy the repository URL

6.5 Push to GitHub

```bash
git remote add origin https://github.com/YOUR_USERNAME/youtube-playlist-shuffler.git
git branch -M main
git push -u origin main
```

6.6 Create Installation Files for Users

Before creating the README, let's set up proper installation files:

Create `requirements.txt`:

```txt
google-api-python-client>=2.70.0
google-auth-oauthlib>=0.5.0
google-auth-httplib2>=0.1.0
requests>=2.28.0
```

Create `setup.py` (Optional - for advanced users):

```python
```

```python
from setuptools import setup, find_packages

setup(
    name="youtube-playlist-shuffler",
    version="1.0.0",
    description="Desktop app to shuffle YouTube playlists",
    author="Your Name",
    python_requires=">=3.8",
    install_requires=[
        "google-api-python-client>=2.70.0",
        "google-auth-oauthlib>=0.5.0",
        "google-auth-httplib2>=0.1.0",
        "requests>=2.28.0"
    ],
    entry_points={
        'console_scripts': [
            'youtube-shuffler=main_gui:main',
        ],
    }
)
```

6.7 Create Comprehensive README.md

🎯 **Challenge Point 4**: Write a README.md file. Here's a template to guide you:

```markdown
```

```
# YouTube Playlist Shuffler

A desktop application that shuffles your YouTube playlists and creates new randomized versions.

## Features
- Connect to your YouTube account securely
- Browse and select your existing playlists
- Shuffle videos in random order
- Create new playlists with shuffled content
- Simple, user-friendly interface

## Screenshots
[Add screenshots of your app here]

## Prerequisites
- Python 3.8 or higher
- Google account with YouTube access
- Internet connection

## Installation Guide

### Step 1: Download the Project
```bash
git clone https://github.com/YOUR_USERNAME/youtube-playlist-shuffler.git
cd youtube-playlist-shuffler
```
```

Step 2: Set Up Python Environment

```bash
# Create virtual environment (recommended)
python -m venv venv

# Activate virtual environment
# On Windows:
venv\Scripts\activate
# On Mac/Linux:
source venv/bin/activate
```

Step 3: Install Dependencies

```bash
```

```
pip install -r requirements.txt
```

Step 4: Set Up YouTube API Access

1. Go to Google Cloud Console
2. Create a new project or select existing one
3. Enable YouTube Data API v3
4. Create OAuth 2.0 credentials (Desktop application)
5. Download the credentials file
6. **Rename it to** `credentials.json`
7. **Place it in the project folder** (same directory as main_gui.py)

Step 5: Run the Application

```bash
python main_gui.py
```

First Time Setup

1. Click "Connect to YouTube"
2. Your browser will open asking for Google account permission
3. Grant access to your YouTube data
4. Return to the app - you should see "✅ Connected"

How to Use

1. **Connect**: Click "Connect to YouTube" and authorize access
2. **Load**: Click "Load Playlists" to see your YouTube playlists
3. **Select**: Choose a playlist from the dropdown
4. **Name**: Enter a name for your new shuffled playlist
5. **Shuffle**: Click "Shuffle & Create Playlist"
6. **Done**: Check your YouTube account for the new playlist!

Project Structure

```
youtube-shuffler/
├── main_gui.py         # Main application GUI
├── playlist_manager.py  # YouTube API operations
├── api_test.py         # Authentication helper
├── credentials.json    # Your API credentials (you add this)
├── token.json          # Auto-generated after first login
├── requirements.txt    # Python dependencies
├── README.md           # This file
└── tests/              # Test files
```

Troubleshooting

"FileNotFoundError: credentials.json"

- Make sure you downloaded OAuth credentials from Google Cloud Console
- Rename the file to exactly `credentials.json`
- Place it in the same folder as main_gui.py

"Authentication failed"

- Check your internet connection
- Verify your Google Cloud project has YouTube Data API enabled
- Make sure you're using OAuth 2.0 credentials (not API key)

"No playlists found"

- Make sure you have playlists in your YouTube account
- Check that you granted all requested permissions during authentication
- Try clicking "Load Playlists" again

GUI freezes during operation

- This is normal for a few seconds during API operations
- Check the log area for progress updates
- If frozen for >30 seconds, restart the app

Development

Running Tests

```bash
```

```
python test_playlist_manager.py
python test_shuffle_create.py
```

Contributing

1. Fork the repository
2. Create a feature branch
3. Make your changes
4. Test thoroughly
5. Submit a pull request

Security Notes

- `credentials.json` contains sensitive API information - never share it
- `token.json` contains your personal access tokens - keep it private
- Both files are in `.gitignore` to prevent accidental commits

License

This project is open source. Feel free to use and modify.

Support

If you encounter issues:

1. Check the Troubleshooting section above
2. Look at the log area in the app for error messages
3. Open an issue on GitHub with detailed error information

Version History

- v1.0.0: Initial release
  - Basic playlist shuffling
  - GUI interface
  - YouTube API integration

```
## Phase 7: Enhancements & Learning

### 7.1 Possible Improvements
- Add playlist preview before shuffling
- Save/load shuffle patterns
- Batch operations on multiple playlists
- Better error handling and user feedback
- Playlist privacy settings

### 7.2 Python Concepts You've Learned
- Object-oriented programming (classes)
- API integration and error handling
- GUI development with tkinter
- Threading for responsive interfaces
- File I/O and JSON handling
- Virtual environments and package management

## Final Testing Checklist
- [ ] Authentication works
- [ ] Playlists load correctly
- [ ] Shuffle and create functionality works
- [ ] Error messages are helpful
- [ ] GUI doesn't freeze during operations
- [ ] Project pushed to GitHub successfully

## Project Structure
```

```
youtube-shuffler/
├── main_gui.py          # Main application
├── playlist_manager.py  # Core functionality
├── api_test.py          # Authentication helper
├── test_*.py            # Test files
├── .gitignore           # Git ignore rules
├── README.md            # Project documentation
├── requirements.txt     # Python dependencies
└── venv/                # Virtual environment (not in git)
```

Congratulations! You've built a complete desktop application with API integration, GUI, and version control. This project demonstrates real-world software development practices and gives you a solid foundation for more complex projects.