Technical Design Document: Twitchy

1. Introduction

This document outlines the technical design and architecture of Twitchy, a self-contained desktop client for viewing Twitch streams and participating in chat. The application is designed for Ubuntu-based systems and prioritizes simplicity and minimal external dependencies by managing its own virtual environment.

The core functionality includes video playback via VLC, real-time chat via a direct IRC connection, and a graphical user interface built with Tkinter. The design emphasizes decoupling the network-intensive chat client from the main UI thread to ensure a responsive user experience.

2. System Architecture

The application operates on a multi-threaded model to separate concerns and prevent the user interface from freezing during network operations.

- **Main Application Thread:** This is the primary thread responsible for the Tkinter event loop. It manages all UI elements, handles user input (e.g., button clicks, window events), and controls the VLC media player instance. It does not perform blocking network I/O.
- **IRC Client Thread:** A secondary, daemonized thread (TwitchIRCClient) is spawned to handle all communications with the Twitch IRC server. This includes connecting, authenticating, sending PONG keep-alive responses, and listening for chat messages.
- **Inter-Thread Communication:** Communication from the IRC thread to the main UI thread is achieved safely using a queue.Queue instance. The IRC client places incoming messages into the queue, and the main thread polls this queue periodically using root.after() to retrieve and display messages without blocking.
- **External Process Execution:** The application relies on the yt-dlp command-line utility to resolve the direct URL for a Twitch stream. This is executed as a sandboxed, short-lived subprocess using subprocess.run().

3. Core Components

The application is composed of several distinct components, each with a specific responsibility.

3.1. Bootstrap and Dependency Management (handle_venv)

This initial execution block ensures the application runs in a controlled environment.

- **Purpose:** To make the script self-contained and portable by automating the creation of a virtual environment and the installation of dependencies.
- **Process Flow:**
  1. The function first checks if the script is already running within a virtual environment (sys.prefix != sys.base_prefix).
  2. If not, it creates a subdirectory named .venv.
  3. It then uses the subprocess module to invoke pip from within the newly created virtual environment to install the required packages listed in the REQUIREMENTS constant (python-vlc, python-dotenv, yt-dlp, Pillow).
  4. Crucially, upon successful installation, it re-launches itself using os.execv, replacing the current process with a new one running under the virtual environment's Python interpreter. This ensures that all subsequent imports (vlc, dotenv, etc.) are resolved correctly.
- **Error Handling:** The function includes specific checks for subprocess.CalledProcessError to provide user-friendly error messages for both virtual environment creation failures and dependency installation failures, including common network-related issues.

3.2. Main Application Class (TwitchApp)

This class encapsulates the entire GUI and application logic.

- **Initialization (__init__)**:
  - Sets up the main Tkinter window, title, and geometry.
  - Binds the <Escape> key for fullscreen toggling and the window close protocol (WM_DELETE_WINDOW) for graceful shutdown.
  - Calls load_config() to populate credentials.
  - Initializes state variables for UI modes (e.g., dark_mode_var).
  - Calls load_emotes() to prepare the local emote rendering system.
  - Calls create_widgets() to build the UI.
  - Starts the non-blocking message queue polling via poll_message_queue().
- **UI Construction (create_widgets)**:

- The UI is structured using Tkinter Frame widgets.
- A PanedWindow is used to create a resizable division between the video frame and the chat frame.
- The video frame is a simple Frame whose window ID is passed to VLC for rendering.
- The chat box is a tk.Text widget configured with a Scrollbar. It is set to a DISABLED state to prevent user text entry, only allowing programmatic insertion.
- tag_configure is used to define text styles for usernames, system messages, and timestamps.

- **Stream Control (load_stream, stop_current_stream)**:
  - load_stream is the primary user-initiated action. It orchestrates stopping any existing stream, fetching the new stream URL via get_stream_url, initializing a new VLC player instance, embedding it in the video frame, and starting a new TwitchIRCClient thread.
  - stop_current_stream provides a clean shutdown sequence for the player and IRC thread, detaching VLC events and joining the thread to ensure resources are released.
- **Event and Message Handling**:
  - poll_message_queue: Uses root.after(100, ...) to schedule itself to run every 100ms. It attempts to get messages from the message_queue in a non-blocking way (get_nowait) and passes them to add_message_to_chat.
  - add_message_to_chat: The final step in the chat data flow. It re-enables the Text widget, inserts the formatted message (with optional timestamp and parsed emotes), disables the widget, and scrolls to the end.
  - handle_stream_end: An event handler bound to the vlc.EventType.MediaPlayerEndReached event. It uses root.after(0, ...) to schedule the cleanup_after_stream_end method on the main thread, ensuring thread safety.

3.3. IRC Client Class (TwitchIRCClient)

This component handles all real-time chat communication in a separate thread.

- **Architecture:** It inherits from threading.Thread and is initialized as a daemon thread, meaning it will not prevent the main application from exiting.
- **Connection Lifecycle (run)**:
  1. Opens a standard TCP socket to irc.chat.twitch.tv on port 6667.
  2. Sends PASS (with the OAuth token) and NICK commands to authenticate.
  3. Sends a JOIN command to enter the specified channel's chat room.
  4. Enters a while loop that continues until its _stop_event is set.
- **Message Processing Loop**:
  - The socket is set to a 1.0-second timeout to prevent the recv call from blocking indefinitely. This allows the loop to periodically check the _stop_event.
  - If a PING message is received from the server, it immediately responds with PONG to maintain the connection.
  - If a PRIVMSG is received, it parses the username and message content from the raw IRC message string. It then formats this into a "username: message" string and puts it onto the shared message_queue.
- **Termination (stop)**: The stop method sets an internal threading.Event, signaling the run loop to terminate gracefully.

3.4. Emote Rendering System

This system allows for the display of local images in place of text within the chat box.

- **Configuration (emotes.json)**: A JSON file maps emote names (e.g., "PogChamp") to their local file paths (e.g., "emotes/pogchamp.png").
- **Loading (load_emotes)**: On startup, the application reads emotes.json. For each entry, it loads the image file using Pillow, resizes it to a standard chat dimension (28x28 pixels), and converts it into a Tkinter.PhotoImage object. These image objects are cached in the self.emote_images dictionary for performance.
- **Parsing (parse_message_with_emotes)**: When a message is to be displayed and emotes are enabled, this method uses a regular expression to find all occurrences of known emote names within the message string. It splits the message into a sequence of text segments and emote names.
- **Rendering (add_message_to_chat)**: The display method iterates through the parsed sequence. Text segments are inserted using chat_box.insert(), while emote names trigger the insertion of the corresponding cached PhotoImage using chat_box.image_create().

4. Configuration and External Files

- **.env file:** A plain text file used for storing sensitive credentials. The application requires:
  - TWITCH_OAUTH_TOKEN: The OAuth token for IRC authentication (e.g., "oauth:xxxxxxxx").
  - TWITCH_NICKNAME: The user's Twitch username.
- **emotes.json:** As described above, maps emote names to file paths.

- **/emotes/ subfolder:** The directory where image files (PNG, GIF, etc.) for local emotes are expected to be stored.

5. Error Handling

The application implements several layers of error handling:

- **Configuration:** Checks for the existence of the .env file and the presence of required variables, displaying a messagebox.showerror on failure.
- **Dependency Installation:** Catches subprocess errors during pip install and provides feedback to the user.
- **Stream Retrieval:** Handles FileNotFoundError (if yt-dlp is not found), subprocess.CalledProcessError (if yt-dlp fails, e.g., stream is offline), and subprocess.TimeoutExpired.
- **VLC Initialization:** A broad try...except block catches potential errors during VLC player instantiation, which commonly occur if VLC is not installed on the host system.
- **IRC Connection:** The TwitchIRCClient wraps its connection logic in a try...except block to catch socket errors or other connection failures, reporting them as a "System:" message in the chat queue.
- **Graceful Shutdown:** The on_closing method ensures that the IRC thread and VLC player are stopped cleanly when the user closes the window, preventing orphaned processes.