System Design Documentation for AutoZap for Chrome

Table of Contents

1. Introduction

1.1 Purpose

The purpose of this document is to provide a comprehensive overview of the system design for AutoZap for Chrome, a tool designed to automate the process of sending WhatsApp messages using the WhatsApp web interface.

1.2 Scope

This system design documentation covers the architecture, components, and functionality of the AutoZap for Chrome application. It includes an overview of the code structure, user interface design, and key functions.

1.3 Definitions, Acronyms, and Abbreviations

- GUI: Graphical User Interface
- CSV: Comma-Separated Values
- QR: Quick Response

1.4 References

- Selenium Documentation: https://www.selenium.dev/documentation/en/
- Tkinter Documentation: https://docs.python.org/3/library/tkinter.html
- PIL Documentation: https://pillow.readthedocs.io/en/stable/index.html

2. System Overview

2.1 System Architecture

AutoZap for Chrome follows a simple client-side architecture. It utilizes the Selenium WebDriver for interacting with the WhatsApp web interface through the Chrome browser. The GUI is built using the Tkinter library for a user-friendly experience.

2.2 Components Overview

The key components of AutoZap for Chrome include:

- Selenium WebDriver: For web automation tasks.
- Tkinter: For building the graphical user interface.
- Threading: Used for running tasks asynchronously.

- CSV and JSON: For handling configuration and input data.
- ImageTk and PIL: For displaying images in the GUI.

2.3 Dependencies

- Selenium
- Tkinter
- Pillow (PIL)
- Chrome WebDriver

3. Design Details

3.1 Load Configuration (load_config function)

This function reads the configuration file (**config.json**) to retrieve essential settings such as the CSV file path and phone number row.

3.2 QR Code Authentication (qrcode_auth function)

Opens the WhatsApp web page and waits for the user to scan the QR code. Handles success and failure scenarios.

3.3 Delete Chrome Profile (delete_chrome_profile function)

Deletes the cached QR code to allow the user to scan a new one.

3.4 Send Message (send_message function)

Uses Selenium to send a WhatsApp message to a specified phone number. Handles various scenarios such as invalid numbers and button availability.

3.5 Log Failures (log_failures function)

Logs phone numbers that failed to receive messages into a CSV file (**phones_that_failed.csv**).

3.6 Main Function (main function)

The main function orchestrates the overall process. It manages the Chrome WebDriver, reads the configuration, sends messages, and handles exceptions.

3.7 Load Frame 1 (load_frame1 function)

Builds the first frame of the GUI, displaying the logo, instructions, and a button to proceed to the next frame.

3.8 Load Frame 2 (load_frame2 function)

Displays options to either scan a new QR code or continue with an existing session.

3.9 Load Frame 3 (load_frame3 function)

Displays the third frame with a progress bar, success and failure counters, and an exit button. It starts a new thread to execute the main process asynchronously.

4. User Interface

4.1 GUI Layout

The GUI is divided into three frames. The first frame prompts the user to start sending messages. The second frame allows the user to choose between scanning a new QR code or continuing with an existing session. The third frame shows the progress and results of the message-sending process.

4.2 Frames and Widgets

- Frame 1: Logo, instructions, and a "SEND MESSAGES" button.
- Frame 2: Logo, options to scan a new QR code or continue, and corresponding buttons.
- Frame 3: Logo, progress bar, success and failure counters, and an exit button.

4.3 Threading for Asynchronous Tasks

The application utilizes threading to run the main process asynchronously, preventing the GUI from freezing during the message-sending operation.

5. Error Handling

5.1 Exception Handling

The application incorporates exception handling to catch and display errors. It uses Tkinter messagebox for user notifications.

5.2 User Notifications

In case of errors or failures, the application provides user-friendly error messages through Tkinter messagebox.

6. Security Considerations

The application interacts with the WhatsApp web interface using Selenium WebDriver. Security considerations include ensuring the safe handling of user data, protecting sensitive information, and adhering to web automation best practices.

7. Future Improvements

Potential future improvements may include:

- Enhanced error logging and reporting.
- Support for additional messaging platforms.
- Improved user interface and customization options.
- Integration with cloud services for better scalability.

This system design documentation provides a comprehensive understanding of the AutoZap for Chrome application, its components, and its functionality. It serves as a reference for developers and users alike.