GROUP 8

TITLE:

“ SMART DESK AI BOT ”

MEMBERS:

1. Muhammad Areeb (24-NTU-CS-FL-1269)
2. Muntaha (24- NTU-CS-FL-1279)
3. Amna Ashraf (24- NTU-CS-FL-1246)
4. Bushra Fatima (24- NTU-CS-FL-1253)
5. Momina Hussain (24- NTU-CS-FL-1264)

❖

Submitted to:

Professor M. Waqar Ahmed

Mam Tahreem Jalil

Documentation

Table of Contents

1: Executive summary

1.1 Introduction

SMARTDESK AI BOT represents a cutting-edge integration of various artificial intelligence services into a single, user-friendly console application. Built in C++, this application serves as a comprehensive AI toolkit that brings together multiple advanced technologies including natural language processing, computer vision, audio processing, and image manipulation.

2. Library Overview and Usage

2.1 CPR (C++ Requests) Library

#include <cpr/cpr.h>

CPR is a modern HTTP library for C++. In our application, it's used for:

- Making HTTP GET and POST requests
- Handling multipart form data
- Managing request headers and parameters
- Processing binary responses

Key features utilized:

cpp

cpr::Response response = cpr::Get(url, parameters, headers) ; ***// GET requests***

cpr::Response response = cpr::Post(url, payload, headers) ; ***// POST requests***

2.2 nlohmann/json Library

cpp

```
#include <nlohmann/json.hpp>
```

This JSON library provides seamless JSON integration in C++. Used for:

- Parsing API responses
- Creating request payloads
- Handling nested JSON structures
- Type-safe JSON manipulation

Example usage:

cpp

```
json payload = {
    {"messages", json::array({
        {{"role", "user"}, {"content", question}}
    })}
};
```

2.3 TinyFileDialogs

cpp

```
#include "tinyfiledialogs/tinyfiledialogs.h"
```

Provides native dialog boxes for file operations:

- File selection dialogs
- Message boxes
- Input dialogs
- Cross-platform compatibility

Example Usage:

```
cpp
const char* filePath = tinyfd_openFileDialog(
    "Select a File",
    "",
    0,
    nullptr,
    nullptr,
    0
);
```

3. Understanding APIs and Request Types

3.1 HTTP Request Methods Used

It is used for retrieving data without modifying server state. In our application it is used in:

- Sentiment analysis queries
- URL shortening
- QR code generation

Example implementation:

```
Response response = Get(
    Url{url},
    Parameters{{"text", input_text}},
    headers
);
```

3.1.2 POST Requests

It is used for sending data to create or update resources. In our project it is used in:

- Image generation
- ChatGPT interactions
- File uploads for processing

Example implementation:

```
Response response = Post(
   Url{url},
   Body{payload.dump()},
   headers
);
```

3.2 API Integration Patterns

Standard headers used across APIs are:

```
Header headers = {
   {"x-rapidapi-key", "YOUR_API_KEY"},
   {"x-rapidapi-host", "api-hostname.com"}
};
```

3.2.2 JSON Payload Structures

Common patterns:

cpp

<u>// ChatGPT API payload</u>

```
json payload = {
   {"messages", json::array({
        {{"role", "user"}, {"content", question}}
    }})},
    {"web_access", "false"}
};
```

// Image generation payload

```
json payload = {
    {"jsonBody", {
        {"function_name", "image_generator"},
        {"type", "image_generation"},
        {"query", text}
    }}
};
```

Architecture

The application follows an object-oriented architecture with each feature implemented as a separate class. Key architectural elements include:

- Main interface class handling user interactions
- Individual service classes for each AI feature
- Common patterns for API communication
- Consistent error handling across components

Each class maintains its own state and handles its specific API communication, making the code modular and maintainable.

External Services

This application relies on several RapidAPI services and other external APIs:

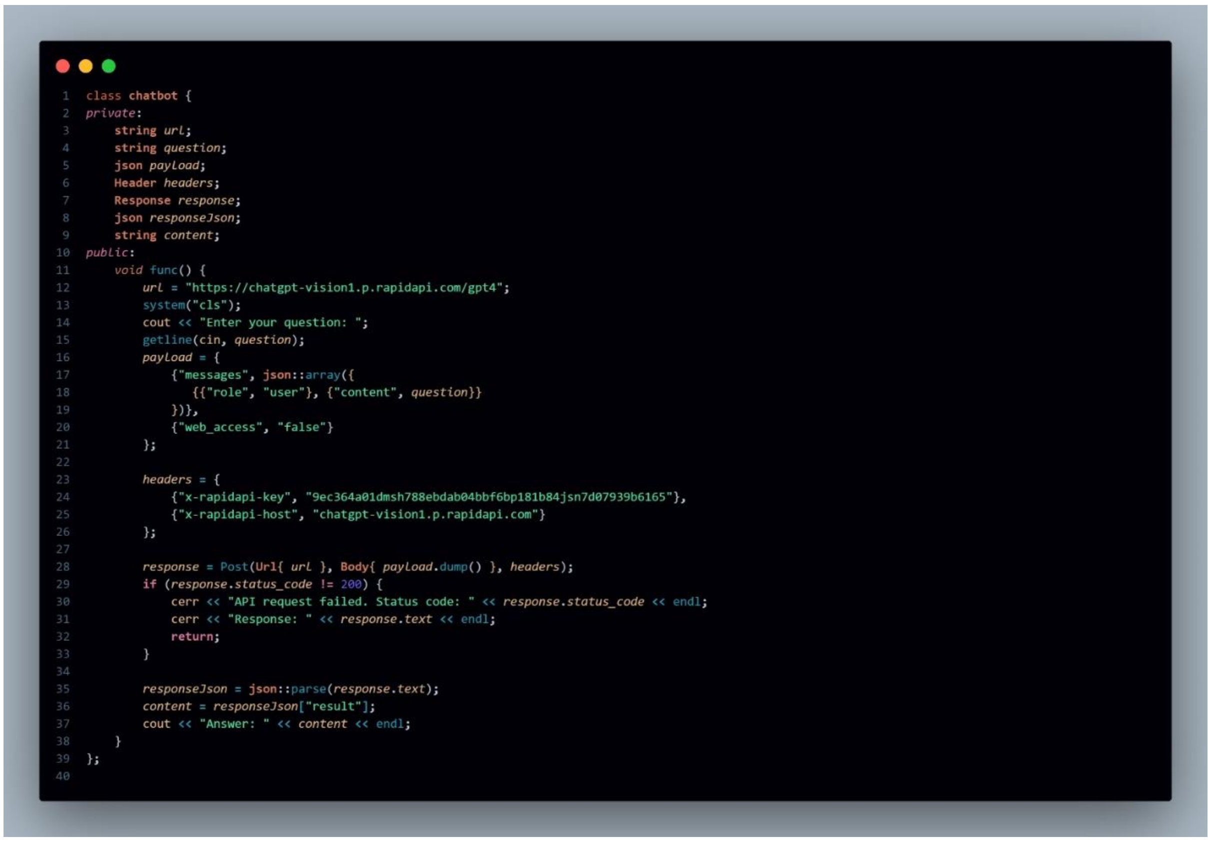
- ChatGPT Vision API
- AI Image Generator API
- AI Background Remover API
- QR Server API
- Text-to-Speech API
- Sentiment Analyzer API
- TinyURL API
- Shazam API
- AI Picture Colorizer API

4. Detailed Feature Documentation

4.1 Chatbot

Implements an AI chatbot using the ChatGPT Vision API.

- Key Methods: func()
- Features: Text-based conversation with AI
- Input: User text input
- Output: AI-generated response

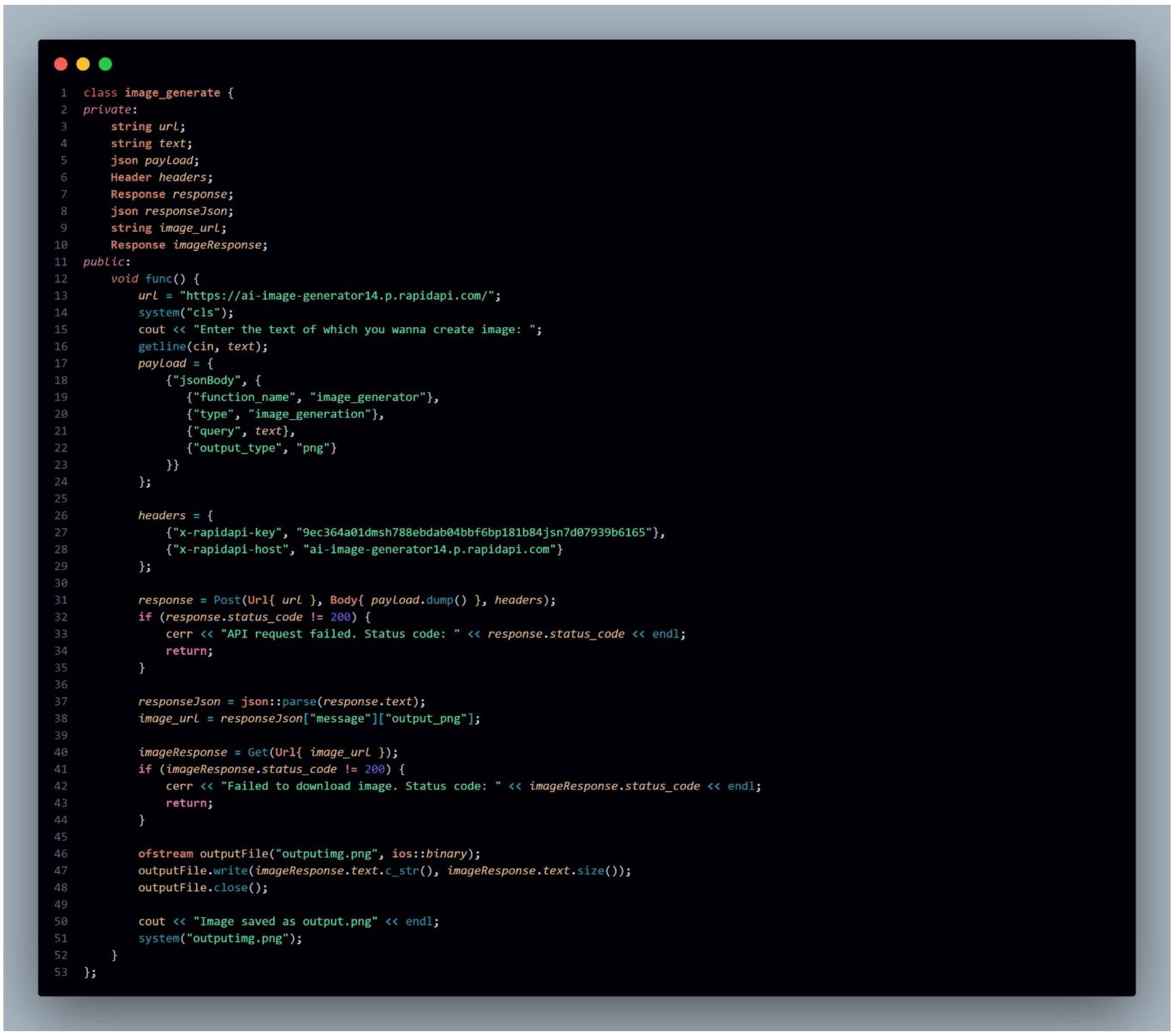


```cpp
class chatbot {
private:
    string url;
    string question;
    json payload;
    Header headers;
    Response response;
    json responseJson;
    string content;
public:
    void func() {
        url = "https://chatgpt-vision1.p.rapidapi.com/gpt4";
        system("cls");
        cout << "Enter your question: ";
        getline(cin, question);
        payload = {
            {"messages", json::array({
               {{"role", "user"}, {"content", question}}
            })},
            {"web_access", "false"}
        };

        headers = {
            {"x-rapidapi-key", "9ec364a01dmsh788ebdab04bbf6bp181b84jsn7d07939b6165"},
            {"x-rapidapi-host", "chatgpt-vision1.p.rapidapi.com"}
        };

        response = Post(Url{ url }, Body{ payload.dump() }, headers);
        if (response.status_code != 200) {
            cerr << "API request failed. Status code: " << response.status_code << endl;
            cerr << "Response: " << response.text << endl;
            return;
        }

        responseJson = json::parse(response.text);
        content = responseJson["result"];
        cout << "Answer: " << content << endl;
    }
};

```

4.2 Image_generate

Handles AI image generation from text descriptions.

- Key Methods: func()
- Features: Converts text descriptions to images
- Input: Text description
- Output: Generated PNG image

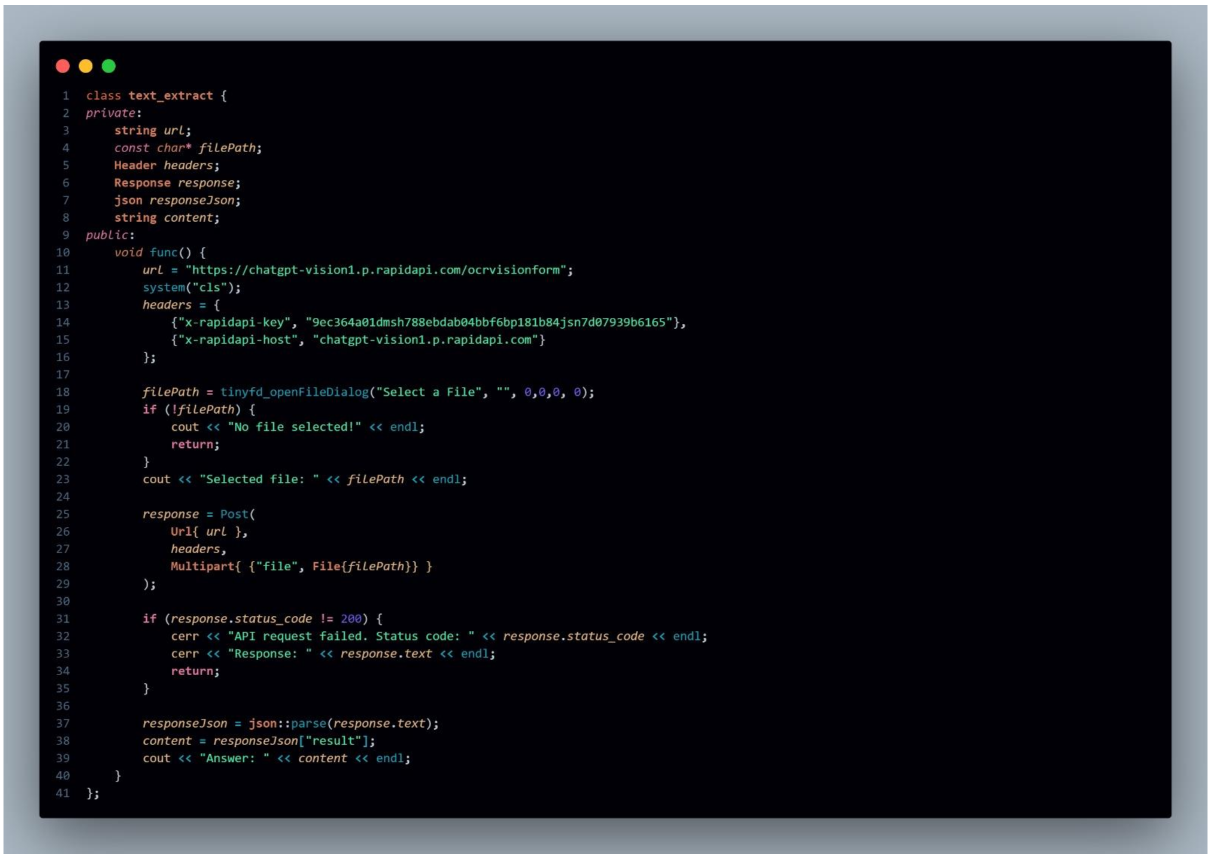


```cpp
class image_generate {
private:
    string url;
    string text;
    json payload;
    Header headers;
    Response response;
    json responseJson;
    string image_url;
    Response imageResponse;
public:
    void func() {
        url = "https://ai-image-generator14.p.rapidapi.com/";
        system("cls");
        cout << "Enter the text of which you wanna create image: ";
        getline(cin, text);
        payload = {
            {"jsonBody", {
               {"function_name", "image_generator"},
               {"type", "image_generation"},
               {"query", text},
               {"output_type", "png"}
            }}
        };

        headers = {
            {"x-rapidapi-key", "9ec364a01dmsh788ebdab04bbf6bp181b84jsn7d07939b6165"},
            {"x-rapidapi-host", "ai-image-generator14.p.rapidapi.com"}
        };

        response = Post(Url{ url }, Body{ payload.dump() }, headers);
        if (response.status_code != 200) {
            cerr << "API request failed. Status code: " << response.status_code << endl;
            return;
        }

        responseJson = json::parse(response.text);
        image_url = responseJson["message"]["output_png"];

        imageResponse = Get(Url{ image_url });
        if (imageResponse.status_code != 200) {
            cerr << "Failed to download image. Status code: " << imageResponse.status_code << endl;
            return;
        }

        ofstream outputFile("outputimg.png", ios::binary);
        outputFile.write(imageResponse.text.c_str(), imageResponse.text.size());
        outputFile.close();

        cout << "Image saved as output.png" << endl;
        system("outputimg.png");
    }
};
```

4.3 Text_extract

Performs OCR on images to extract text.

- Key Methods: func()
- Features: Image-to-text conversion
- Input: Image file
- Output: Extracted text

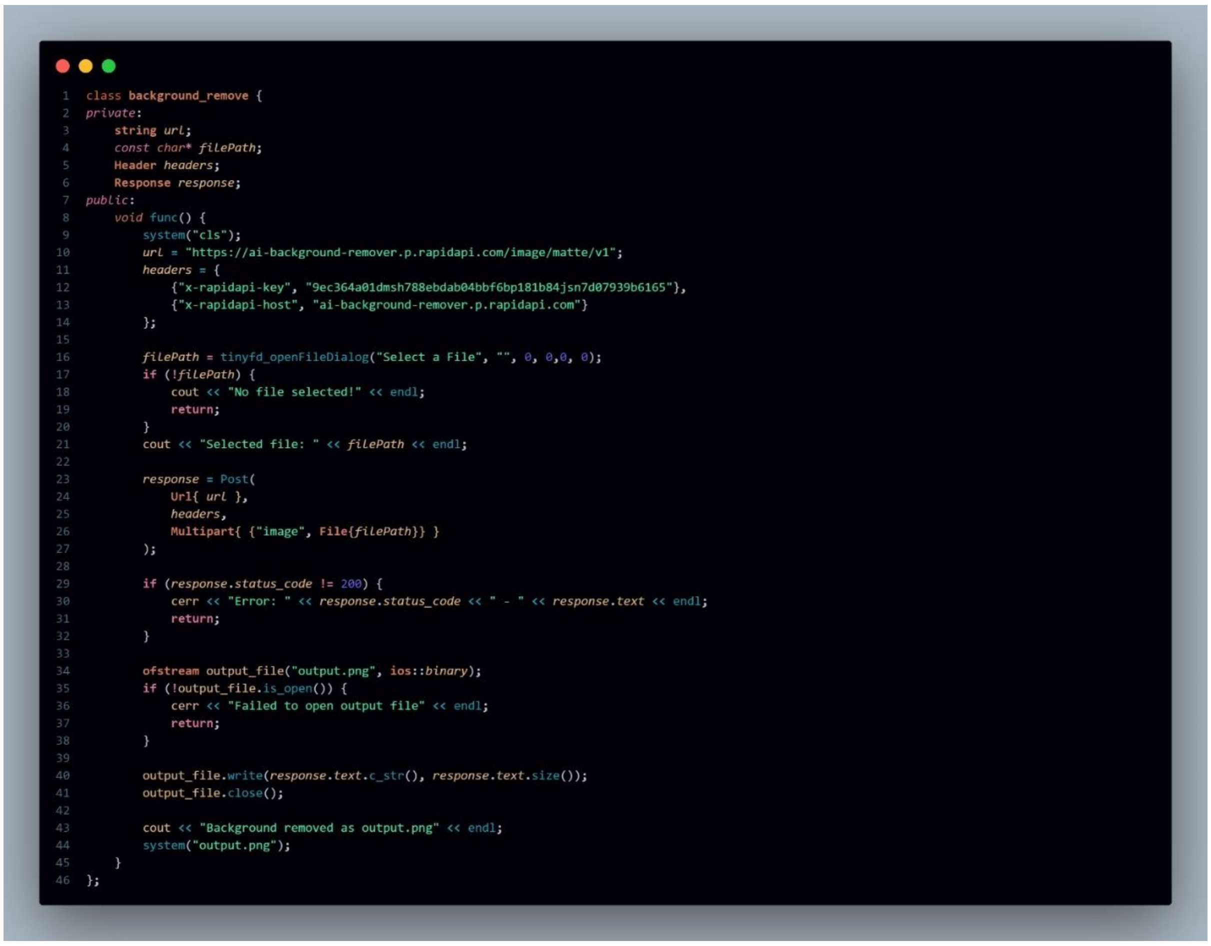


```cpp
class text_extract {
private:
    string url;
    const char* filePath;
    Header headers;
    Response response;
    json responseJson;
    string content;
public:
    void func() {
        url = "https://chatgpt-vision1.p.rapidapi.com/ocrvisionform";
        system("cls");
        headers = {
            {"x-rapidapi-key", "9ec364a01dmsh788ebdab04bbf6bp181b84jsn7d07939b6165"},
            {"x-rapidapi-host", "chatgpt-vision1.p.rapidapi.com"}
        };

        filePath = tinyfd_openFileDialog("Select a File", "", 0,0,0, 0);
        if (!filePath) {
            cout << "No file selected!" << endl;
            return;
        }
        cout << "Selected file: " << filePath << endl;

        response = Post(
            Url{ url },
            headers,
            Multipart{ {"file", File{filePath}} }
        );

        if (response.status_code != 200) {
            cerr << "API request failed. Status code: " << response.status_code << endl;
            cerr << "Response: " << response.text << endl;
            return;
        }

        responseJson = json::parse(response.text);
        content = responseJson["result"];
        cout << "Answer: " << content << endl;
    }
};
```

4.4 Background_remove

Removes backgrounds from images using AI.

- Key Methods: func()
- Features: Automatic background removal
- Input: Image file
- Output: Processed image without background

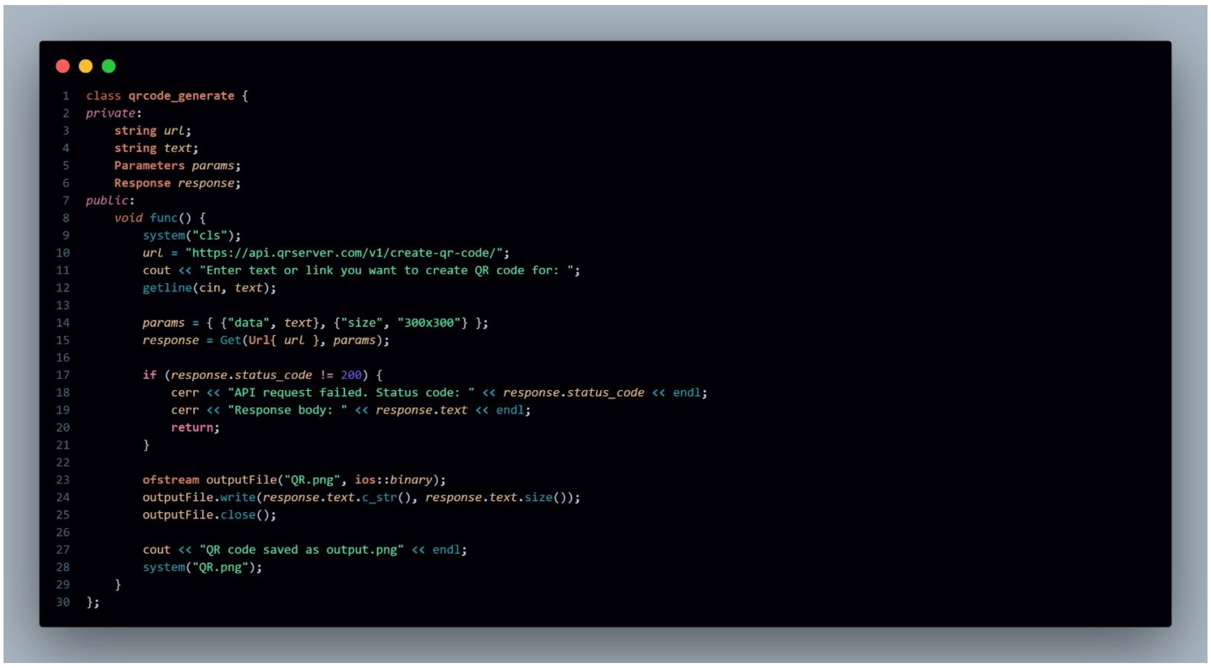


```cpp
class background_remove {
private:
    string url;
    const char* filePath;
    Header headers;
    Response response;
public:
    void func() {
        system("cls");
        url = "https://ai-background-remover.p.rapidapi.com/image/matte/v1";
        headers = {
            {"x-rapidapi-key", "9ec364a01dmsh788ebdab04bbf6bp181b84jsn7d07939b6165"},
            {"x-rapidapi-host", "ai-background-remover.p.rapidapi.com"}
        };

        filePath = tinyfd_openFileDialog("Select a File", "", 0, 0,0, 0);
        if (!filePath) {
            cout << "No file selected!" << endl;
            return;
        }
        cout << "Selected file: " << filePath << endl;

        response = Post(
            Url{ url },
            headers,
            Multipart{ {"image", File{filePath}} }
        );

        if (response.status_code != 200) {
            cerr << "Error: " << response.status_code << " - " << response.text << endl;
            return;
        }

        ofstream output_file("output.png", ios::binary);
        if (!output_file.is_open()) {
            cerr << "Failed to open output file" << endl;
            return;
        }

        output_file.write(response.text.c_str(), response.text.size());
        output_file.close();

        cout << "Background removed as output.png" << endl;
        system("output.png");
    }
};
```

4.5 Qrcode_generate

Generates QR codes from text or URLs.

- Key Methods: func()
- Features: QR code generation
- Input: Text or URL
- Output: QR code image

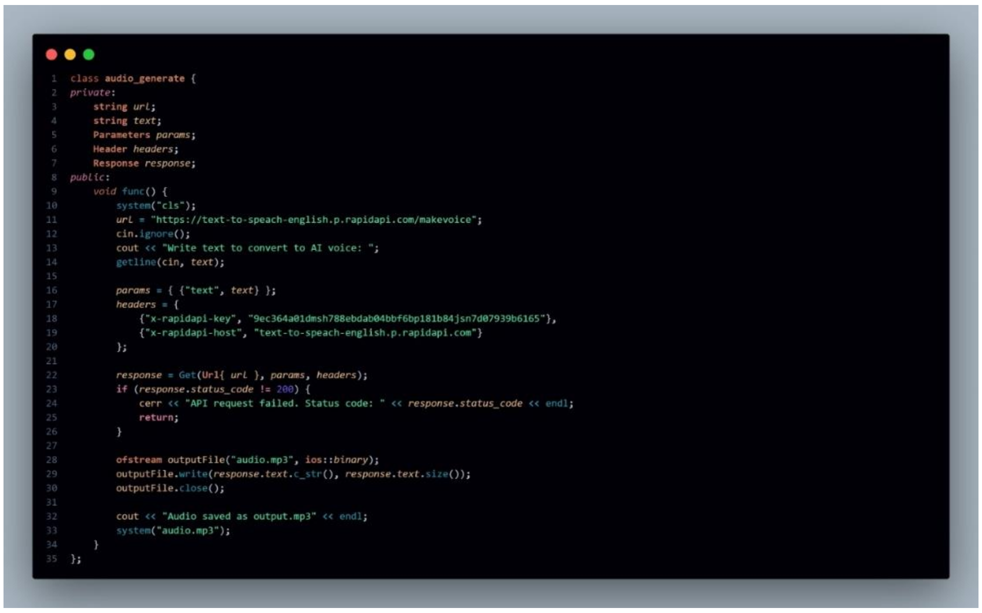


```cpp
class qrcode_generate {
private:
    string url;
    string text;
    Parameters params;
    Response response;
public:
    void func() {
        system("cls");
        url = "https://api.qrserver.com/v1/create-qr-code/";
        cout << "Enter text or link you want to create QR code for: ";
        getline(cin, text);

        params = { {"data", text}, {"size", "300x300"} };
        response = Get(Url{ url }, params);

        if (response.status_code != 200) {
            cerr << "API request failed. Status code: " << response.status_code << endl;
            cerr << "Response body: " << response.text << endl;
            return;
        }

        ofstream outputFile("QR.png", ios::binary);
        outputFile.write(response.text.c_str(), response.text.size());
        outputFile.close();

        cout << "QR code saved as output.png" << endl;
        system("QR.png");
    }
};
```

4.6 Audio_generate

Converts text to speech using AI.

- Key Methods: func()
- Features: Text-to-speech conversion
- Input: Text
- Output: MP3 audio file

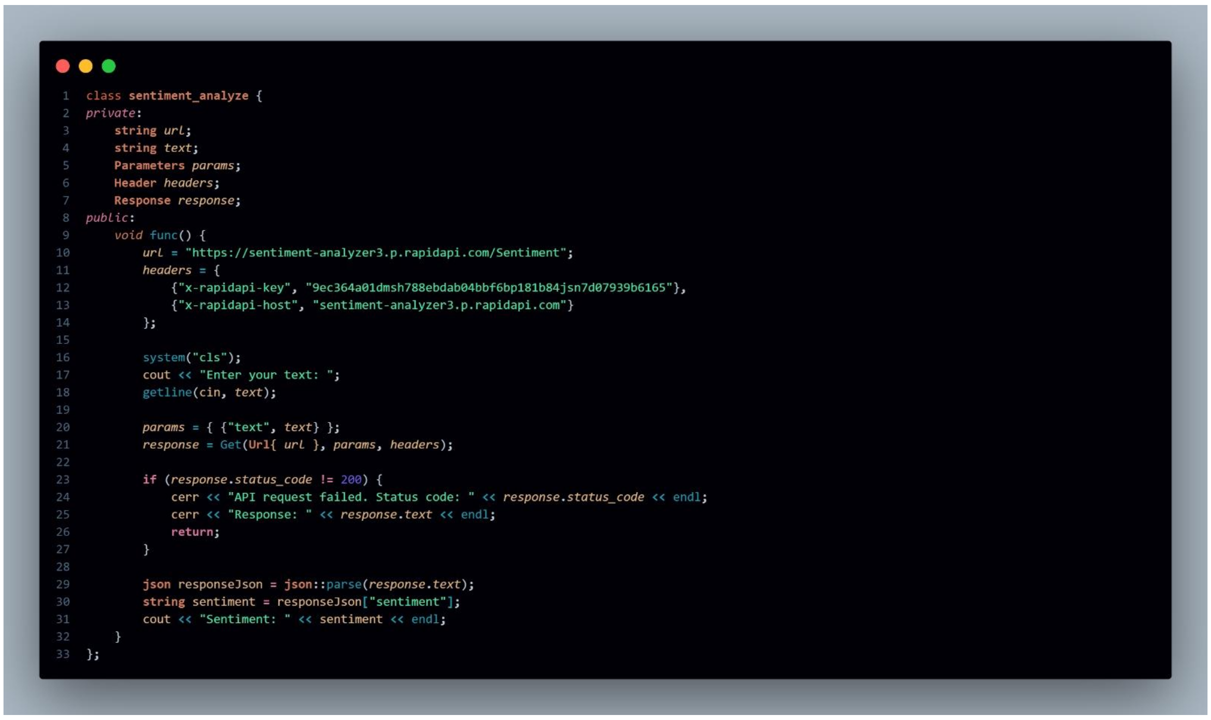


```cpp
class audio_generate {
private:
    string url;
    string text;
    Parameters params;
    Header headers;
    Response response;
public:
    void func() {
        system("cls");
        url = "https://text-to-speach-english.p.rapidapi.com/makevoice";
        cin.ignore();
        cout << "Write text to convert to AI voice: ";
        getline(cin, text);

        params = { {"text", text} };
        headers = {
            {"x-rapidapi-key", "9ec364a01dmsh788ebdab04bbf6bp181b84jsn7d07939b6165"},
            {"x-rapidapi-host", "text-to-speach-english.p.rapidapi.com"}
        };

        response = Get(Url{ url }, params, headers);
        if (response.status_code != 200) {
            cerr << "API request failed. Status code: " << response.status_code << endl;
            return;
        }

        ofstream outputFile("audio.mp3", ios::binary);
        outputFile.write(response.text.c_str(), response.text.size());
        outputFile.close();

        cout << "Audio saved as output.mp3" << endl;
        system("audio.mp3");
    }
};
```

4.7 Sentiment_analyze

Analyzes the sentiment of text input.

- Key Methods: func()
- Features: Text sentiment analysis
- Input: Text
- Output: Sentiment classification

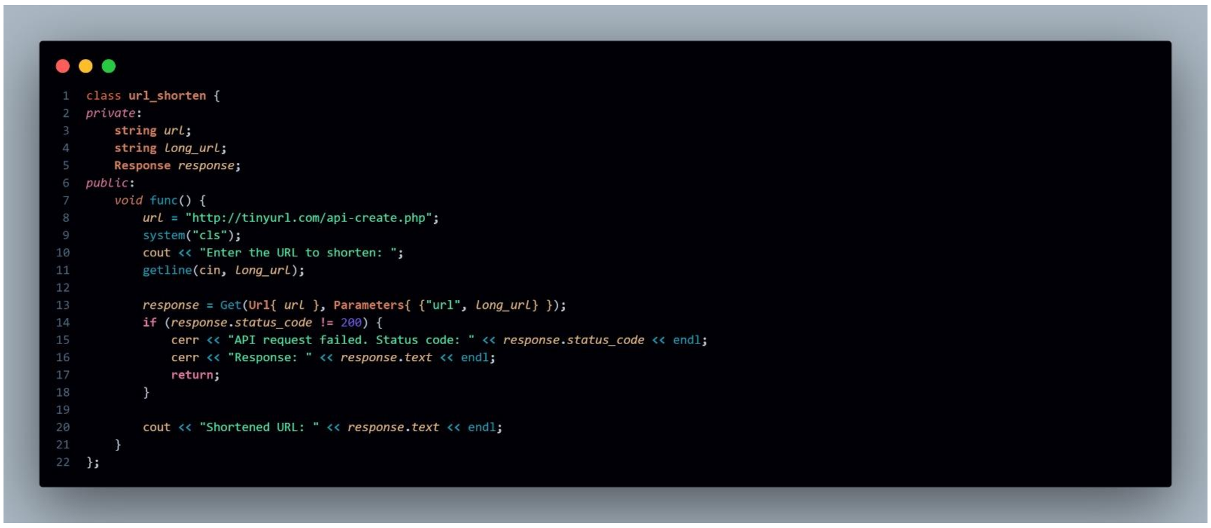


```cpp
class sentiment_analyze {
private:
    string url;
    string text;
    Parameters params;
    Header headers;
    Response response;
public:
    void func() {
        url = "https://sentiment-analyzer3.p.rapidapi.com/Sentiment";
        headers = {
            {"x-rapidapi-key", "9ec364a01dmsh788ebdab04bbf6bp181b84jsn7d07939b6165"},
            {"x-rapidapi-host", "sentiment-analyzer3.p.rapidapi.com"}
        };

        system("cls");
        cout << "Enter your text: ";
        getline(cin, text);

        params = { {"text", text} };
        response = Get(Url{ url }, params, headers);

        if (response.status_code != 200) {
            cerr << "API request failed. Status code: " << response.status_code << endl;
            cerr << "Response: " << response.text << endl;
            return;
        }

        json responseJson = json::parse(response.text);
        string sentiment = responseJson["sentiment"];
        cout << "Sentiment: " << sentiment << endl;
    }
};
```

4.8 Url_shorten

Shortens URLs using the TinyURL service.

- Key Methods: func()
- Features: URL shortening
- Input: Long URL
- Output: Shortened URL

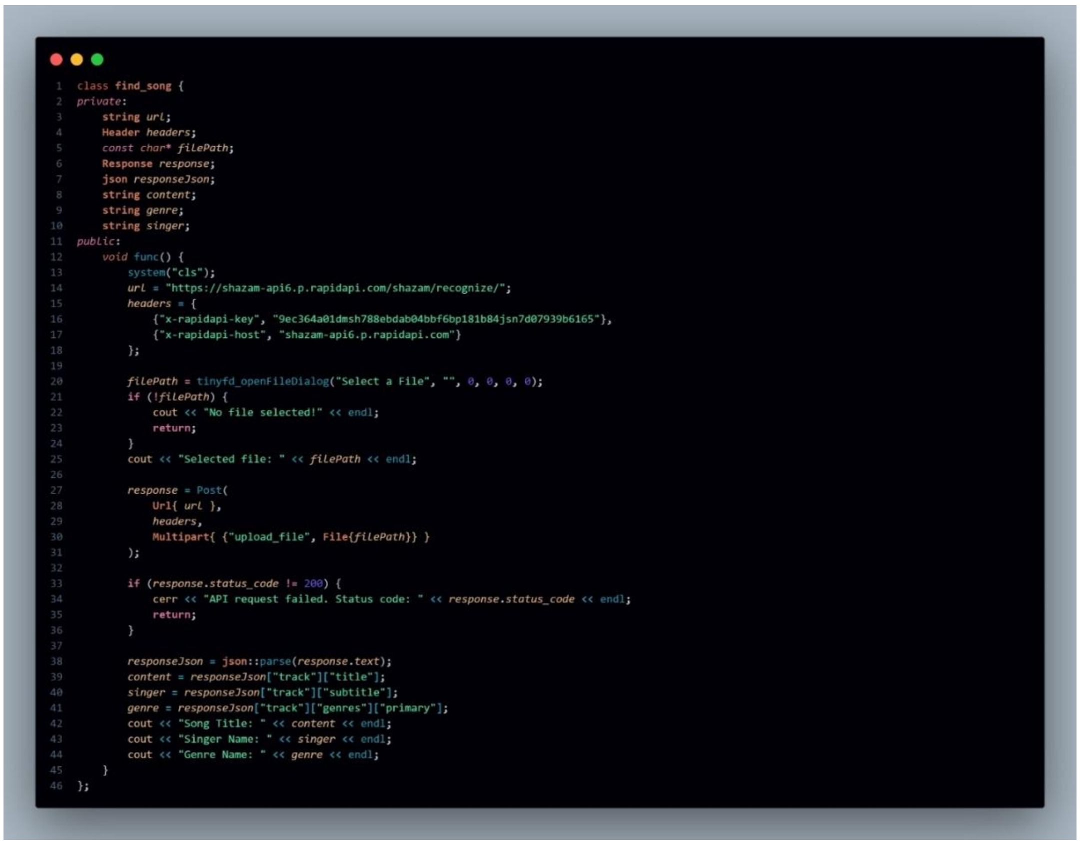


```cpp
class url_shorten {
private:
    string url;
    string long_url;
    Response response;
public:
    void func() {
        url = "http://tinyurl.com/api-create.php";
        system("cls");
        cout << "Enter the URL to shorten: ";
        getline(cin, long_url);

        response = Get(Url{ url }, Parameters{ {"url", long_url} });
        if (response.status_code != 200) {
            cerr << "API request failed. Status code: " << response.status_code << endl;
            cerr << "Response: " << response.text << endl;
            return;
        }

        cout << "Shortened URL: " << response.text << endl;
    }
};
```

4.9 Find_song

Identifies songs from audio samples.

- Key Methods: func()
- Features: Audio recognition
- Input: Audio file
- Output: Song details (title, artist, genre)

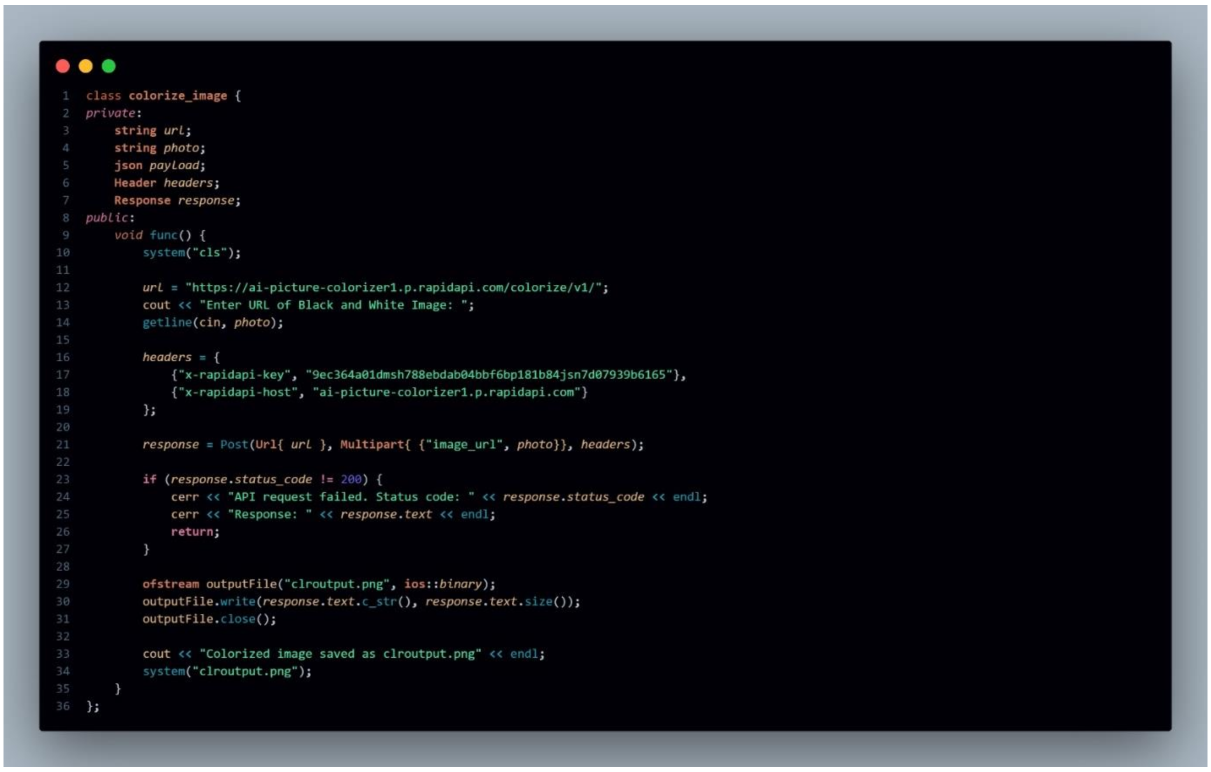


```cpp
class find_song {
private:
    string url;
    Header headers;
    const char* filePath;
    Response response;
    json responseJson;
    string content;
    string genre;
    string singer;
public:
    void func() {
        system("cls");
        url = "https://shazam-api6.p.rapidapi.com/shazam/recognize/";
        headers = {
            {"x-rapidapi-key", "9ec364a01dmsh788ebdab04bbf6bp181b84jsn7d07939b6165"},
            {"x-rapidapi-host", "shazam-api6.p.rapidapi.com"}
        };

        filePath = tinyfd_openFileDialog("Select a File", "", 0, 0, 0, 0);
        if (!filePath) {
            cout << "No file selected!" << endl;
            return;
        }
        cout << "Selected file: " << filePath << endl;

        response = Post(
            Url{ url },
            headers,
            Multipart{ {"upload_file", File{filePath}} }
        );

        if (response.status_code != 200) {
            cerr << "API request failed. Status code: " << response.status_code << endl;
            return;
        }

        responseJson = json::parse(response.text);
        content = responseJson["track"]["title"];
        singer = responseJson["track"]["subtitle"];
        genre = responseJson["track"]["genres"]["primary"];
        cout << "Song Title: " << content << endl;
        cout << "Singer Name: " << singer << endl;
        cout << "Genre Name: " << genre << endl;
    }
};
```

4.10 Colorize_image

Colorizes black and white images using AI.

- Key Methods: func()
- Features: Image colorization
- Input: B&W image URL
- Output: Colorized image



```cpp
class colorize_image {
private:
    string url;
    string photo;
    json payload;
    Header headers;
    Response response;
public:
    void func() {
        system("cls");

        url = "https://ai-picture-colorizer1.p.rapidapi.com/colorize/v1/";
        cout << "Enter URL of Black and White Image: ";
        getline(cin, photo);

        headers = {
            {"x-rapidapi-key", "9ec364a01dmsh788ebdab04bbf6bp181b84jsn7d07939b6165"},
            {"x-rapidapi-host", "ai-picture-colorizer1.p.rapidapi.com"}
        };

        response = Post(Url{ url }, Multipart{ {"image_url", photo}}, headers);

        if (response.status_code != 200) {
            cerr << "API request failed. Status code: " << response.status_code << endl;
            cerr << "Response: " << response.text << endl;
            return;
        }

        ofstream outputFile("clroutput.png", ios::binary);
        outputFile.write(response.text.c_str(), response.text.size());
        outputFile.close();

        cout << "Colorized image saved as clroutput.png" << endl;
        system("clroutput.png");
    }
};
```

<u>5. Error Handling and Troubleshooting</u>

API Communication Errors

if (response.status_code != 200) {

cerr << "API request failed. Status code: " << response.status_code << endl;

```
    cerr << "Response: " << response.text << endl;
    return;
}
```

File Operation Errors

```
if (!outputFile.is_open()) {
    cerr << "Failed to open output file" << endl;
    return;
}
```

Error Prevention Strategies

1. Input Validation
2. Response Checking
3. File System Verification
4. Network Status Verification

6.Best Practices and Usage Guidelines

1. Ensure stable internet connection
2. Verify API key validity
3. Monitor API usage limits
4. Regular error log checking

<u>8. Installation and Setup</u>

Prerequisites

1. C++ Development Environment
2. Required Libraries
3. API Access Keys
4. System Requirements

Installation Steps

1. Library Installation
2. Project Configuration
3. API Key Setup
4. Compilation Process

<u>9. Maintenance and Updates</u>

Regular Maintenance Tasks

1. API Key Rotation
2. Library Updates
3. Error Log Review
4. Performance Monitoring

Update Procedures

1. Code Backup
2. Library Update Process
3. API Version Checking
4. Testing Procedures