**Introduction**

A Hotel Management System (HMS) is a software application specifically designed to streamline and automate the operations of a hotel or hospitality business. It encompasses a wide range of functionalities to manage various aspects of hotel operations, including reservations, guest check-in/check-out, room assignment, billing and invoicing, inventory management, staff scheduling, and reporting.

Key features of a Hotel Management System typically include:

1. **Reservation Management:** Allows hotel staff to manage room reservations, including checking availability, making bookings, and handling cancellations.
2. **Guest Check-in/Check-out:** Facilitates the smooth process of guest arrivals and departures, including room assignment, key card issuance, and payment processing.
3. **Room Management:** Tracks the status of rooms (clean, dirty, occupied, vacant) and manages room inventory.
4. **Billing and Invoicing:** Handles guest billing, including room charges, additional services (like meals or laundry), and generates invoices for guests.
5. **Point of Sale (POS) Integration:** Integrates with the hotel's POS system to manage food and beverage sales, gift shop purchases, and other retail transactions.
6. **Inventory Management:** Manages hotel inventory such as linens, toiletries, and other supplies, ensuring that stock levels are maintained.
7. **Reporting and Analytics:** Provides insights into hotel performance through reports on occupancy rates, revenue, and other key metrics.
8. **Security and Access Control:** Ensures data security and controls access to sensitive information, such as guest details and financial transactions.
9. **Staff Management:** Manages staff schedules, tasks, and performance to ensure efficient operations and guest satisfaction.
10. **Integration with Online Booking Platforms:** Integrates with online travel agencies (OTAs) and the hotel's website for seamless online bookings and updates to room availability.

**Objective**

1. **Efficient Operations:** Streamlining day-to-day operations such as reservations, check-ins, check-outs, room assignments, and housekeeping tasks to improve overall efficiency and productivity.
2. **Enhanced Guest Experience:** Providing a seamless and enjoyable experience for guests from the moment they make a reservation to the time they check out, including personalized services and efficient handling of requests.
3. **Revenue Management:** Maximizing revenue through effective pricing strategies, room inventory management, and upselling of additional services such as dining, spa treatments, or tours.
4. **Resource Optimization:** Efficiently managing resources such as rooms, staff, inventory, and facilities to minimize waste and reduce costs.
5. **Data Management and Analysis:** Collecting and analyzing data on guest preferences, booking patterns, and operational performance to make informed decisions and improve business strategies.
6. **Compliance and Security:** Ensuring compliance with regulations and industry standards related to data security, guest privacy, and other legal requirements.
7. **Integration and Automation:** Integrating with other systems such as point-of-sale (POS) systems, online booking platforms, and accounting software to automate processes and improve accuracy.
8. **Staff Management:** Optimizing staff schedules, training, and performance to provide excellent service while controlling labor costs.
9. **Marketing and Guest Relations:** Leveraging guest data to personalize marketing efforts, improve guest loyalty, and enhance the hotel's reputation.
10. **Sustainability:** Implementing practices that promote environmental sustainability and responsible tourism, aligning with modern trends and customer preferences.

Requirements

**To run this program, you'll need the following:**

* C++ compiler (e.g., g++)
* libcurl library (for making HTTP requests)
* jsoncpp library (for parsing JSON)
* Weatherstack API key (sign up at https://weatherstack.com/ to obtain a paid API key)
* Technologies Used C++
* libcurl (CURL library)
* jsoncpp (JSON library)

System Architecture

The program follows a simple linear architecture, with the following **components:**

* User input for the city name.
* Constructing the API URL with the user's API key and city.
* Making an HTTP request to the Weatherstack API.
* Parsing the JSON response.
* Displaying the weather data.

Installation

* Install a C++ compiler if not already installed (e.g., g++).
* Install the libcurl library (instructions may vary based on your OS).
* Install the jsoncpp library (instructions may vary based on your OS).
* Obtain a Weatherstack API key by signing up at <https://weatherstack.com/>.

Usage

* Compile the program using your C++ compiler.
* Run the compiled binary.
* Enter the name of the city for which you want to retrieve weather data.

Features

* Fetches and displays current weather data for a specified city.
* Handles API requests and responses using libcurl.
* Parses JSON responses using jsoncpp.
* Displays information such as city name, country, temperature, wind speed, and humidity.

**Code Structure**

The code is structured as follows:

* Initialization of libcurl and setting up options.
* Making an API request.
* Handling API errors.
* Parsing the JSON response.
* Displaying the weather data.

**Code execution order**

* **Library Includes:** The code includes necessary libraries, including <iostream>, <curl/curl.h> for making HTTP requests using libcurl, and <json/json.h> for parsing JSON data.
* **API Key Setup:**

It defines a constant string API\_KEY that holds an API key for accessing weather data.

* **Callback Function:**

It defines a callback function” **WriteCallback**” that is used to receive and append data received from the HTTP request.

* **Main Function:**

The main function is where the program starts its execution.

* **CURL Initialization:**

It initializes libcurl using curl\_easy\_init() and checks if initialization was successful.

* **User Input:**

It prompts the user to enter a city name and stores it in the city variable.

* **API URL Construction:**

It constructs the API URL using the entered city name and the API key.

* **CURL Options Configuration:**

It sets various libcurl options using curl\_easy\_setopt. These options include the URL to fetch, the path to a CA certificate bundle (cacert.pem) for secure connections, and the callback function to handle the response data.

* **HTTP Request Execution:**

It executes the HTTP request using curl\_easy\_perform(). If the request fails, it prints an error message and cleans up libcurl resources.

* **Response Processing:**

If the request succeeds, it prints the received weather data.

* **JSON Parsing:**

It uses the JsonCpp library to parse the JSON response data into a JSON object (root) for easy access to specific fields.

* **Display Weather Data:**

It extracts and displays specific weather information such as city name, country, temperature, wind speed, and humidity from the JSON object.

* **Program Termination:**

Finally, the program cleans up libcurl resources using curl\_easy\_cleanup() and returns 0 to indicate successful execution.

Testing

Testing can be performed by running the program with different city names and verifying that it retrieves and displays weather data correctly.

Performance

The performance of this program mainly depends on the response time of the Weatherstack API and the network conditions. It's designed for simple usage and may not be suitable for high-performance applications.

Future Enhancements

**Possible future enhancements for this program could include:**

Adding error handling for invalid API responses.

Implementing more advanced weather data display options.

Supporting multiple API providers.

Contributing

Contributions to the code are welcome through pull requests on the project's GitHub repository (if available).

License

This program is provided under an open-source license (mention the specific license if applicable).

Conclusions

This program serves as a basic example of how to interact with RESTful APIs and parse JSON responses in C++. It can be a starting point for more complex weather-related applications or similar API integrations.

Acknowledgments

This C++ program was developed by”***Aryan gupta*** “for educational purposes. It demonstrates how to use the cURL library to make HTTP requests to an external API and how to parse JSON responses using the JsonCpp library.

The code makes use of the following libraries:

- cURL (libcurl): https://curl.se/

- JsonCpp: https://github.com/open-source-parsers/jsoncpp

The Weatherstack API is used to retrieve weather information based on user input, and it requires an API key for access. The API key used in this code is provided by Weatherstack and should be kept secure.

This code is meant to serve as a learning resource for those interested in working with web APIs, handling HTTP requests, and parsing JSON data in C++.

Please note that the code may require adjustments or updates to work with the latest versions of the libraries or changes in the Weatherstack API.

ARYAN GUPTA

28/08/2023

45 Days Daily Report

**Week 1 :**

Day 1 :

* Code implementation started.
* Basic error handling for libcurl implemented.
* No major issues encountered.

Day 2-7 :

* Continued to use the code daily to fetch weather data for single city.
* Observed consistent functionality and performance.
* Identified a need for better error messages to improve user experience. Implemented more descriptive error messages.

**Week 2 :**

Day 8-14 :

* Further usage of the code for daily weather data retrieval.
* Discovered that some city names with spaces were causing API query issues. Improved user input validation to handle spaces in city names.
* Code robustness improved with better error handling and logging.

**Week 3 :**

Day 15-21:

* Continued daily usage of the code.
* Noted a potential performance issue when fetching data for multiple cities in a single run. Investigated and optimized the HTTP request mechanism to reduce latency.
* Improved readability and added comments for better code maintainability.

**Week 4 :**

Day 22-28 :

* Ongoing daily usage without any significant issues.
* Considered implementing a feature to store historical weather data for future reference.
* Documented the code, including usage instructions and potential improvements for the future.

**Week 5 :**

Day 29-35:

* Continued daily use and monitoring.
* Conducted a comprehensive code review, examining code quality, adherence to best practices, and potential security vulnerabilities.
* Ensured that the code was thoroughly documented for future maintainers.
* Considered implementing automated testing to ensure continued reliability.
* Completed the 5-week daily report, concluding that the code remains reliable and efficient for fetching weather data.

**WEEK 6 :**

**Day 35-42:**

* Conducted a comprehensive code review.
* Ensured adherence to best practices and coding standards.
* Code quality improved.
* Considered potential optimizations or future enhancements.
* Documented the overall 6-week experience.

**Summary:**

Over the course of 6 weeks, the code for fetching weather data has been tested daily and has remained stable and reliable.

Error handling and user-friendliness were improved, making the code more robust.

The code is ready for continued use and potential future enhancements.

Documented the 6-week journey to track improvements and stability.

This extended testing and reporting period should help ensure the code's long-term reliability and maintainability. Additionally, it provides an opportunity to identify and address any issues that may arise over a more extended period of use