Using curl

sexta-feira, 25 de abril de 2025 19:15

- curl is used toget and query APIs from outside the project
- If ill use only one function it's fine, but i may have many others.... so it's better to create na abstraction

- First of all i need to understand the base structure to execute a query in na endpoint
  - Pass a callback function to write the results
  - pass an endpoint do execute the query
  - Would be nice to use that wrapper
  - cleanup in the destructor...
    - So when i'll need it... better to create it inside na scope and leave the destructor handle the garbage collector.
    - This is too simple to take the risk of manage memory myself
- Th plan is to create na interface to encapsulate the httpclient requests
  - Define na interface
  - The code will depend on the interface
  - We can mock fo runit testing
  - and may use diferent http libs

✅ **Interface-based dependency injection**
✅ **Low coupling between DownloadFiles and Curl**
✅ **Easier testing (mocking IHttpClient)**
✅ **Separation of concerns** (DownloadFiles does not care how HTTP works, only that it works)

🧠 Why Use an Interface?

Reason	Explanation
Testability	You can create a **fake** or **mock** HTTP client for unit testing without making real HTTP calls.
Flexibility	You can change your HTTP backend (like from libcurl to Boost.Beast or cpp-httplib) easily.
Separation of Concerns	Business logic shouldn't know about low-level HTTP details.
Dependency Inversion Principle	Code depends on **abstractions**, not concrete implementations (important SOLID principle).

🧠 What you have now

You created a design like this:

cpp

```cpp
class IHttpClient {
public:
    virtual ~IHttpClient() = default;
    virtual bool getResponseFromUrl(const std::string& url, std::string& response) = 0;
};
```

Concrete implementation:

cpp

```cpp
class CurlHttpClient : public IHttpClient {
public:
    CurlHttpClient(); // Constructor
    ~CurlHttpClient(); // Destructor
    bool getResponseFromUrl(const std::string& url, std::string& response) override;
};
```

🛠️ How you use it

Inside your `DownloadFiles` class:

cpp

```cpp
bool DownloadFiles::isValidUrl(IHttpClient& httpClient)
{
#ifdef UNIT_TEST
    return true;
#endif

    std::string response;
    return httpClient.getResponseFromUrl(m_originalURL, response);
}
```

✅ Notice: **Pass IHttpClient by reference**, not by value (avoids slicing and copies).

When you use it normally (in production code), you **instantiate the concrete** `CurlHttpClient` and **pass it**:

cpp

```cpp
CurlHttpClient realHttpClient;
DownloadFiles downloader;
bool valid = downloader.isValidUrl(realHttpClient);
```

In **unit tests**, you will **inject a MockHttpClient** instead:

cpp

```cpp
class MockHttpClient : public IHttpClient {
public:
    bool getResponseFromUrl(const std::string& url, std::string& response) override {
        response = "fake response"; // or whatever you need for the test
        return true;
    }
};
```

In your unit test code:

cpp

```cpp
MockHttpClient mockHttpClient;
DownloadFiles downloader;
EXPECT_TRUE(downloader.isValidUrl(mockHttpClient));
```

When you instantiate, prefer **dependency injection by constructor** instead of passing it on every method. Example:

cpp

```cpp
class DownloadFiles {
private:
    IHttpClient& m_httpClient;
public:
    DownloadFiles(IHttpClient& httpClient) : m_httpClient(httpClient) {}

    bool isValidUrl();
};
```

Then use:

cpp

```cpp
CurlHttpClient realHttpClient;
DownloadFiles downloader(realHttpClient);
downloader.isValidUrl();
```

➡️ This way `DownloadFiles` always has a valid HTTP client and you don't have to pass it manually to each method. Cleaner!

🔥 Diagram

sql

```sql
+---------------+
|  IHttpClient    | <--- Interface
+---------------+
        ^
        |
+-----------------+         +------------------+
| CurlHttpClient    |         | MockHttpClient     |
| (Real HTTP)       |         | (Fake for tests)   |
+-----------------+         +------------------+
        ^
        |
+---------------------+
| GitHubDownloader      |
| (Business logic)      |
+---------------------+
```