HTTP Client C & REST API

Duțu Alin Călin

CONTENTS

1 Introduction **2**

1.1 Context 2

1.2 Objectives 2

2 Proposed Solution **3**

2.1 Architecture 3

2.2 Communication 4

2.2.1 Endpoints 4

2.2.2 JWT Token 6

3 Implementation Details **7**

3.1 Proposed implementation 7

3.2 Commands 7

3.3 Project files 9

3.4 Implementation description 9

3.5 Main Flow 11

3.6 Testing 11

4 Conclusions **12**

Bibliography **13**

1 INTRODUCTION

1.1 Context

Web apps represent a meaningful percentage of modern applications. Most of these applications, despite their complexity, are designed based on the client-server model, a classic design model. To understand how the client-server model works, it is required to comprehend how the HTTP protocol works.

1.2 Objectives

The purpose of this program is to create a C HTTP client that can interact with a server using the REST API. The following objectives have to be covered after implementing the client:

- Understanding of the mechanism of communication used by HTTP in the background
- The interactions with a REST API server
- Understanding the concepts that are frequently used in the web environment such as JSON, session, and JWT tokens
- The usage of external libraries to manipulate JSON objects, which are crucial for communication with a REST API server.

2 PROPOSED SOLUTION

This project aims for the implementation of a web client that can successfully interact with a server. Despite that, the de facto technology used for designing these clients is the triad: HTML, CSS, and JavaScript, a more familiar approach using C language was chosen to fathom the concepts and to closely approach the protocol.

2.1 Architecture

Looking from a general point of view, there can be 2 entities that perform different actions and interactions to ensure communication:

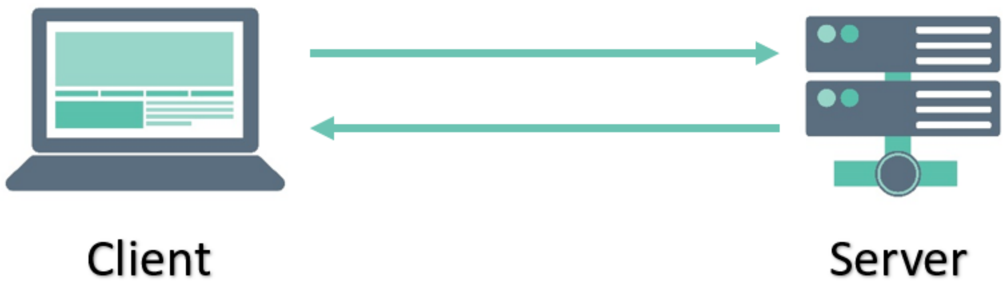

Figure 1: Application architecture [1]

The server exposes an API (Application Programmable Interface) of type REST (Representational State Transfer). This is like a black box that exposes a series of entries represented by HTTP routes called endpoints. Following the HTTP requests, the server executes an action. From the project's perspective, the server simulates an online library and it is already implemented.

The client is a program developed in C that accepts commands from the keyboard and sends requests to the server depending on the command type. This program aims to work as closely as a UI interface for the virtual library.

[1]`https://www.flaticon.com/icons`. Icons created by RawPixel

2.2 Communication

The REST API protocol secures the connection between the client and the server. The client will send all the commands via HTTP requests, while the server will process them accordingly as soon as its endpoints receive them.

2.2.1 Endpoints

Account registry

- Access Route: POST /api/v1/tema/auth/register
- Payload type: application/json
- Payload:

```
{
    "username": String,
    "password": String
}
```

- Returns an error if the username already exists

Authentication

- Access Route: POST /api/v1/tema/auth/login
- Payload type: application/json
- Payload:

```
{
    "username": String,
    "password": String
}
```

- Returns cookie session
- Returns an error message if the credentials are not valid

Access request in the library

- Access Route: GET /api/v1/tema/library/access
- It has to be demonstrated that the user is authenticated
- Returns a JWT token, which demonstrates the access to the library
- Returns an error message if the user doesn't demonstrate that he is authenticated

Visualizing a summary of information about all the books

- Access Route: GET /api/v1/tema/library/books
- It has to be demonstrated that the user has access to the library
- Returns a list of JSON objects:

```
[{
    id: Number,
    title: String
}]
```

- Returns an error message if the user doesn't demonstrate that he has access

Visualizing details about a book

- Access Route: GET /api/v1/tema/library/books/:bookId. ":bookId" will be replaced with the ID of the book
- It has to be demonstrated that the user is authenticated
- Returns a JSON object:

```
{
    "id": Number,
    "title": String,
    "author": String,
    "publisher": String,
    "genre": String,
    "page_count": Number
}
```

- Returns an error message if the user doesn't demonstrate that he has access
- Returns an error message if the book ID is invalid

Adding a book

- Access Route: POST /api/v1/tema/library/books
- Payload Type: application/json
- It has to be demonstrated that the user has access to the library
- Returns a JSON object:

```
{
    "title": String,
    "author": String,
    "genre": String,
    "page_count": Number,
    "publisher": String
}
```

- Returns an error message if the user doesn't demonstrate that he has access
- Returns an error message if either the added information is incomplete or it doesn't respect the format

Deleting a book

- Access Route: DELETE /api/v1/tema/library/books/:bookId. ":bookId" will be replaced with the ID of the book
- It has to be demonstrated that the user has access to the library
- Returns an error message if the user doesn't demonstrate that he has access
- Returns an error message if the request is invalid

Logout

- Access Route: GET /api/v1/tema/auth/logout
- It has to be demonstrated that the user is authenticated
- Returns an error message if the user doesn't demonstrate that he is authenticated

2.2.2 JWT Token

JWT tokens are used to ensure communication integrity between clients and servers. To create one, the information is converted into binary and signed to make sure that an attacker won't modify the information from packets.

To send the token to the server, it is required to add the word **Bearer** and set it in the **Authorization** header.

3 IMPLEMENTATION DETAILS

3.1 Proposed implementation

The proposed solution mainly consists of an application implemented in C++ that must interpret commands from the keyboard to interact with the server. When the client receives 1 command, it will create the required JSON objects, send the request, and then show the response. The process shall repeat until the exit command is called.

3.2 Commands

The input for each command looks as follows:

Register

```
register
username=<username>
password=<password>
```

Login

```
login
username=<username>
password=<password>
```

Enter library

```
enter_library
```

Get books

```
get_books
```

Get book

```
get_book
id=<id>
```

Add book

```
add_book
title=<title>
author=<author>
genre=<genre>
publisher=<publisher>
page_count=<page_count>
```

Delete book

```
delete_book
id=<id>
```

Logout

```
logout
```

Exit

```
exit
```

3.3 Project files

1. Files from PCOM labs [1]
    - buffer.c
    - buffer.h
    - helpers.c
    - helpers.h
    - request.c (modified version)
    - request.h (modified version)
2. Files from the Parson open-source library [2]
    - parson.c
    - parson.h
3. Other files
    - client.c
    - Makefile
    - README

3.4 Implementation description

Main homework implementation is found in the client.c, request.c and request.h files.

1. **Request.c and Request.h**

    Files were taken from PCOM labs. Modifications regarding header and cookie processing were added to adapt to the purpose of this project. An additional function has been added to process DELETE requests which is close to a POST function as it adds URL, Host, and the necessary headers and a newline to the message. In request.h function signatures are found, while implementation of these signatures is found in the request.c.

2. **Client.c**

    Here resides the main implementation. Firstly, the host is computed by merging the server IP with the connection port which will let the user add new commands until it chooses to use the exit command, which will close the connection and exit the program. During this phase, the user can add commands to create HTTP requests for the server to execute and the following functions will be used:

    - register - Called when the user prompts "register". The function will create a POST request by creating a JSON object using the provided username and password, send the request to the server, and show the response as soon as the client receives it.

- login - Called when the user prompts "login". It creates a POST request in the same manner as the register command with the notable difference of saving the session cookie in case of a successful login.

- enter_library - Called when user prompts "enter_library". It will send a simple GET request containing the session cookie. Upon receiving a successful response, the JWT authorization token provided by the server will be received.

- get_books - Called when user prompts "get_books". It will process a GET request with the saved authorization token and show the list of books available in a JSON format upon success.

- get_book - Called when "get_book" is prompted. It will process a GET request with the JWT token as a header and the book ID provided by the user in the URL request. The result upon success is the book information in JSON format.

- add_book - Called when "add_book" is prompted. It will process a POST request with the authorization token as a header and the information about the new book provided by the user in JSON format and show the response message.

- delete_book - Called when "delete_book" is prompted. It will process a DELETE request with the authorization token as a header and the book ID provided by the user in the request URL. Upon receiving the response from the server, its message will be shown.

- logout - Called when "logout" is prompted. It will process a GET request which contains the session cookie. Upon success. The tokens will be deleted from the program's memory and a corresponding message will be shown.

For executing multiple commands simultaneously, due to the technical difficulties met, it has been decided upon the end of processing a request, regardless of the output, to close and reopen the client socket. The exception to the rule is represented by the exit command which will only close the socket.

The Parson library made a major contribution to the process of serializing and deserializing the objects used in the client's requests and server's replies due to its converting functions that can be easily integrated into the program.

3.5 Main Flow

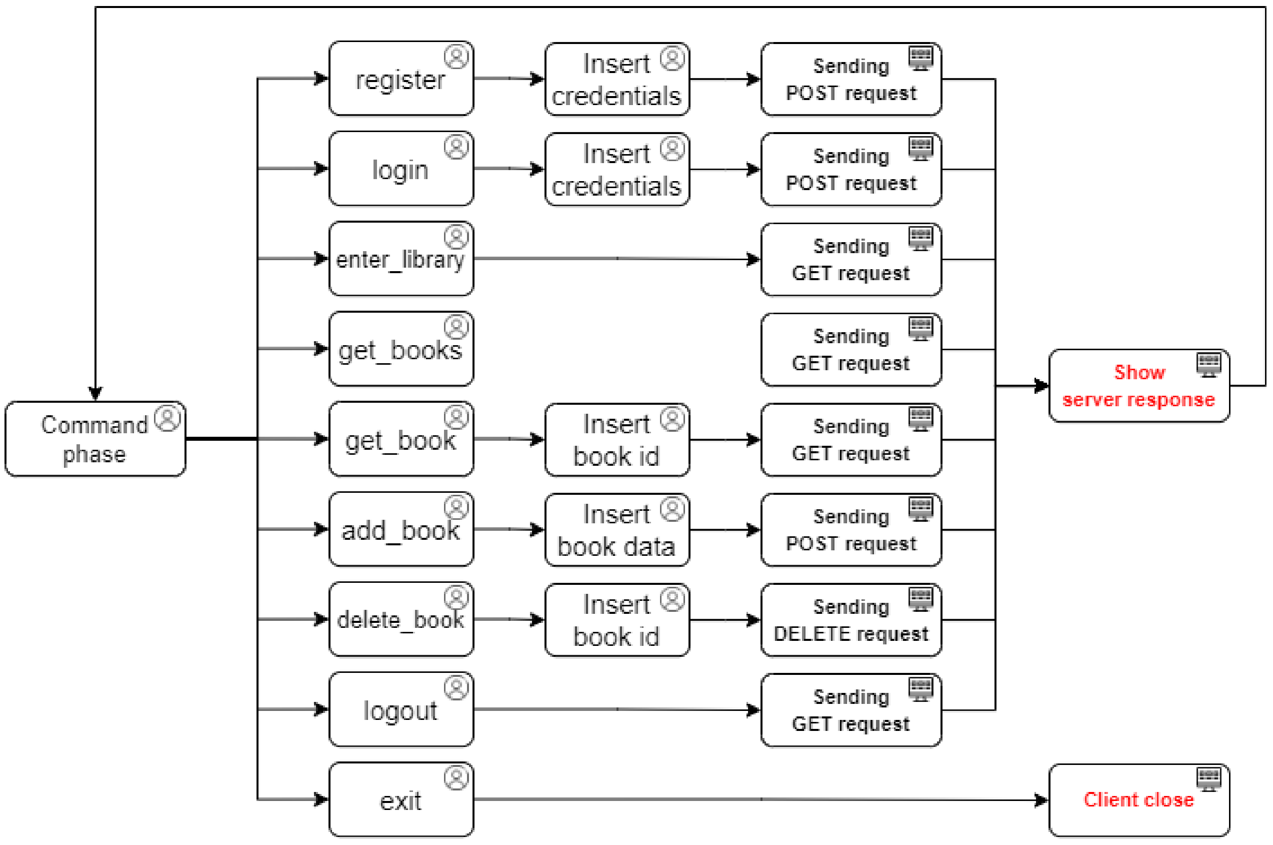

Figure 2: Main Flow diagram

3.6 Testing

For testing the proposed solution, the following methods have been used:

- Postman - Used for testing the requests provided by the client
- Manual testing - Used for testing the program's behavior in all the possible conditions.

4 CONCLUSIONS

In conclusion, this project proposes an HTTP client implemented in C that can interact with a server using the REST API to understand the basic mechanics used by HTTP by implementing communication with a server based on REST API and how is data represented in the communication between these entities from a client's perspective.

BIBLIOGRAPHY

[1] PCom team @ Polytechnic University of Bucharest. Laboratory 10 PCom. https://ocw.cs.pub.ro/courses/pc/laboratoare/10. Latest access: 03.03.2024.

[2] kgabis Github users: disconnec3d. Parson Library. https://github.com/kgabis/parson. Latest access: 03.03.2024.