

SOFTWARE REQUIREMENTS FOR REUNIONLOG

REUNION

L^AT_EX

Main Programmer and lead Designer - NICKGISMOKATO

Design contributor - BJ

Design contributor & tester - CASPER

Written by
Nick LAURSEN

Preface

This is the documentation of the requirements we want to follow when creating the program REUNIONLOG. We will be following the SOLID principles. This software is a program meant for the guild REUNION in the game WORLD OF WARCRAFT. This software will use the API from WARCRAFTLOGS.

Most of these requirements have been gathered from multiple months of pre-gathering data from the WARCRAFTLOGS API. This has been done by creating a *proof-of-concept* program with PYTHON.

This software is under the MIT LICENSE. Read LICENSE for more information.

Contents

Contents	2
1 WarcraftLogs and the API	4
1.1 Website	4
1.1.1 Overall form	4
1.1.2 Specific to our needs	4
1.2 Documentation for the API	5
1.2.1 GraphQL	5
1.2.2 Limitation of the API	5
1.3 Integration from the API to C#	6
1.3.1 Authentication	6
1.3.2 Query Strings	7
1.3.3 Data extracted from the API	7
2 Our software main requirements	8
2.1 Authentication	8
2.2 Events	8
2.3 Query Strings	8
2.4 .CSV file	8
3 Flow of ReunionLog	9
3.1 Simple Overview	9
3.2 Needs for each main Requirements	9
4 Responsibilities for our requirements	10
4.1 Responsibilities	10
4.2 UML Diagram	10

<i>CONTENTS</i>	3
A Figures	11
A.1 Pictures	11
Bibliography	12

Chapter 1

WarcraftLogs and the API

Abstract

This chapter will go through WARCRAFTLOGS and the API correlating. For this chapter we will not go through the actual documentation but rather give a short refer to the documentation and lay out the most important aspect from the site and the API

1.1 Website

The website for WARCRAFTLOGS¹ is a popular website used by guilds to gather data to one single site. This is done through multiple addons and their own in-house software.

1.1.1 Overall form

The data that can be collected is both in the form of GUILD data, RAID data, DUNGEON data, CHARACTER data and much more. These can be accessed for all through the website. This can be done by everyone anonymously.

The website uses GraphQL to display most of their data. Both in tables both also in graphs and tables containing graphics. This gives an easier overview for most users. They also have some options you can choose for the specific data you want to be showed.

A downside to this approach is that a lot of the specific options is not showed. One could reason the "simpler" design is because they want all users to use their website, no matter their technical background.

1.1.2 Specific to our needs

What described in **Section 1.1.1** sound really great and reasonable useful for most users case. This is indeed the case for most users, but if you want the information not available on the site or you want another way to represent the data, then the site is not for you. This is why we will be using the data given to us by the API.

The only need we have of the website is to check if the information we get is also the information displayed on the site.

¹<https://www.warcraftlogs.com/>

1.2 Documentation for the API

The API and its documentation is, for a lack of a better word, idiotic made. There exists two API's. Version 1 and version 2. We will be using the latter. This version has "better" documentation than its counterpart and uses OAUTH 2.0 for its API authentication. The documentation can be found two places. For authentication documentation you can find it at Warcraftlogs 2023 and you can find the actual command call documentation at WarcraftLogs and GraphQL 2023.

1.2.1 GraphQL

The first thing we should worry about is the authentication. As mentioned in the **Preface** we have already made successful connection, therefore this will be discussed later.

GraphQL is the schema of how to make calls to the API. This is done by sending you authentication and a "Query" call. This is in simple terms just a string with specific data. This data is both used to tell WARCRAFTLOGS server what you would like to receive about also where in the schema the server would have to lookup this data. Clearly there is a need to create these query strings.

Therefore we will be using GraphQL library from GRAPHQL-DOTNET. More information can be found at GraphQL 2023. If you were to look at WarcraftLogs and GraphQL 2023 you would find there is a lot of commands you can send. Therefore it is important to create an object which can be agile and create all the necessary strings which we will be using.

1.2.2 Limitation of the API

When it comes to the API a lot of limits start to show. Some in the documentation and some in the actual API itself. Let us start talking about the API itself. When on a free tier, i.e not subscribed to WARCRAFTLOGS website, you can at maximum make 3.600 calls to their server pr. hour. This inherently doesn't sound that bad, but you will later see in depths why this is a bad system.

I will agree that this stops most novice DDOS attacks to their server, but since the introduction of the second version of their API, multiple clients can easily be set up and could potentially help in the DDOS attacks. Let us now explain why this rate limit is bad when you want to make a software that pulls a great amount of data.

Let us say you had a guild that was raiding and you both raided MYTHIC and HEROIC during the same log. Let us furthermore assume you want the data for when they die and how many pulls they were on, but only for the mythic. By scouring through their schema documentation (WarcraftLogs and GraphQL 2023), you find the Report object inside the documentation. This is essentially what you want. So you create a query string with report as first argument and then you scour through the documentation and find the event option which deliver a ReportEventPaginator which is just a list event in the log file. This gets returned as data in form of a JSON string. Let us see the documentation for it:

"A set of paginated report events, filterable via arguments like type, source, target, ability, etc. This data is not considered frozen, and it can change without notice. Use at your own risk." (WarcraftLogs and GraphQL 2023)

This object has an argument you can filter from called difficulty. The documentation for this is as follow:

”**difficulty**: Optional. Whether or not to filter the fights to a specific difficulty. By default all fights are included.” (WarcraftLogs and GraphQL 2023)

This is brilliant. We can now filter our `Event` to `mythic` only. Later down the line is argue that `difficulty` is an integer value. Great so now we just need to figure out what integer value. This is the first hurdle we found when trying this. No actual documentation is given for the arguments, except if it is an object or enum type which they have created. So we set up in PYTHON a script to run through with value n where $n \in \{-1000, -999, \dots, 999, 1000\}$. Then we checked if there was any difference between the JSON strings we received and we got that there was no difference for any n value.

Clearly any user would be confused. Why would an API give you an option and then not document said option and actually do nothing with this option. Your first instinct would now be to contact WARCRAFTLOGS. So you write them an email and their response they send you back can be seen in fig. A.1. So you go to their discord find the thread about the API and ask your question. The intention of this documentation is not to out anyone from their support team and therefore I will not share the conversation I had. But trust us when we say that if it was probably the most useless information and support we have ever received.

So how did we fix it, you may ask? We looked at the documentation and again read what was written under the `difficulty` argument. The `difficulty` argument says it filters fights. So we then took a look at the `ReportFight` object. We tested this and sure enough it worked. So to simplify what we now have to do if we wish to get the event data only for MYTHIC. First we need to get all the fight data that had that difficulty i.e. the `fightIDs`. This is done with the `ReportFlight` object. Then we have to sort our `ReportEventPaginator` to the `fightIDs`. Now we have used two calls of our 3.600 rate limit.

In our demo in PYTHON it became clear that we also had to pull a lot more data than just fights i.e. pulls and so on. We now have to pull every single fight, every single name from that fight and the event itself. This gives us a rough estimate:

$$l \cdot (a_{\text{Fights}} + a_{\text{Event}} + a_{\text{name}} \cdot p),$$

where p is the amount of players in any fights and l is the amount of different logs we want to pull data from. Since we cannot get `fightIDs` whilst also getting the event to only get data from the mythics, we would need to do this over two separate calls and also we cannot pull names just from the event log the way we want it, so we have to do it separately.

This is probably the most irritating limit of the API and therefore we would need to make sure our software can handle this limit in case our request limit gets met.

1.3 Integration from the API to C#

As discussed in 1.2 it is clear that we would need many objects to handle everything involving WARCRAFTLOGS API and its authentication. Clearly we would need objects handling our query strings, *all of them*, handling of the authentication and most importantly handling the data we receive after we have made a call, i.e. the response.

1.3.1 Authentication

Creating an authentication for our software is actually quite easy. We don't need any libraries for the creation of the code or the requester. The only thing we need is a user's `client_id` and `client_secret`. The `tokenURL` is a constant i.e.

"*https://www.warcraftlogs.com/oauth/token*". Everything else is user input. This is also important but will be dealt with differently.

We will need the user to create the `client_id` and `client_secret` which we will save as local data for further use. This will be done at first time use of the software.

1.3.2 Query Strings

When dealing with

1.3.3 Data extracted from the API

Chapter 2

Our software main requirements

Abstract

2.1 Authentication

2.2 Events

2.3 Query Strings

2.4 `.CSV` file

Chapter 3

Flow of ReunionLog

Abstract

3.1 Simple Overview

3.2 Needs for each main Requirements

Chapter 4

Responsibilities for our requirements

Abstract

4.1 Responsibilities

4.2 UML Diagram

Appendix A

Figures

A.1 Pictures

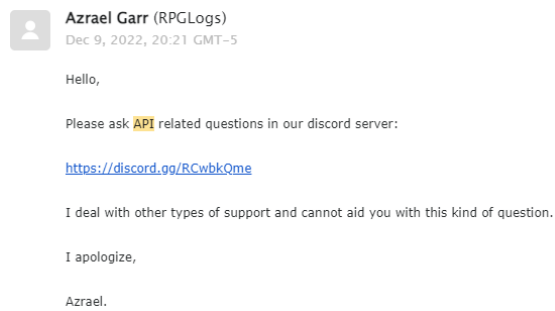


Figure A.1: Mail received from the support team at WARCRAFTLOGS

Bibliography

GraphQL. 2023. “graphql-dotnet.” <https://github.com/graphql-dotnet/graphql-dotnet>.

Warcraftlogs. 2023. “Docs.” <https://www.warcraftlogs.com/api/docs>.

WarcraftLogs and GraphQL. 2023. “GraphQL schema documentation.”
<https://www.warcraftlogs.com/v2-api-docs/warcraft/>.