// football-data.org

Resources

I have only 5 resources to offer, but in the end that’s all you need to built something like this. Well, not quite in a rush, but you won’t need anything other than the tools these resources and it’s subresources provide.
The API follows best practices unfortunately most widely missed in other APIs. I don’t want to sound snooty, but that’s my sad experience while implementing countless (so called RESTful) APIs at my day-job or for personal use within the last decade.

Resource design

The resources of the football-data API are designed to be logical for human minds to reason about but also clean and straightforward to use within your source code.

URI rules

Resources and subresources within URIs are all written lower case. Filters on the other hand use camelCase instead and last but not least enums are all uppercase.

All resources are only accessible using their plural, thus a resource by default responds with its' list representation.
A list resource URI

    https://api.football-data.org/v4/teams

Adding an id to the endpoint gives access to one particular resource:
A single resource URI

    https://api.football-data.org/v4/teams/19

I often use different resource representations, in case they are embedded into list resources. I don’t think every information is useful in every resource (representation), which is why I often limit embedded objects to a subset of their attributes. I know this is opinionated and not ideal for strongly typed languages, but I also know there are ways around that and I think the advantages outbalance here.

Responses

Resources are represented as JSON objects. List representations typically hold some meta-information on top to give you more information about it’s response. Let’s illustrate that with an example. Let’s call the Matches Subresource for FC Bayern München.

Response for https://api.football-data.org/v4/teams/5/matches

```
{
   "filters": {
      "competitions": "CL,BL1,DFB",
      "permission": "TIER_THREE",
      "limit": 100
   },
   "resultSet": {
      "count": 46,
      "competitions": "BL1,DFB,CL",
      "first": "2021-08-13",
      "last": "2022-05-14",
      "played": 46,
      "wins": 22,
      "draws": 7,
      "losses": 7
   },
   "matches": [ ... ]
}
```

On top you see the filters that were applied to the request. Filters might be applied automatically by the API (that is: setting defaults). They can also be passed in (manually) by query parameters to set or overwrite the values being used.
As we didn’t specify any in the example, we only see default filters for that resource, e.g. the limitation of 100 items (which we could overwrite by suffixing "?limit=150" to the request URI).
The resultSet node gives some information about the boundaries of the match list that follows as well as some basic aggregation data. Last but not least the list of match items follows.

Requesting a Resource

For a first glimpse you can use your webbrowser to browse the API. To make the responses human-readable, install a browser-plugin that beautifies JSON output, or better yet use Postman or Kongs Insomnia (not to be mixed up with the one of Faithless). Of course you can also use curl or Powershells' Invoke-WebRequest to natively fetch resources, depending on your platform.

To implement the API there are lots of libraries and plugins that ease implementing RESTful APIs. I personally can recommend Unirest-Java Library also by Kong for Groovy/Java, the awesome Requests lib for Python and Guzzle for PHP. But most probably you already got your favorite HTTP library at hand.

The API follows the Query-string composition standard. So if you want to use Filters read more about query definition on wikipedia.

> Please implement smart requests and always try to think about a good tradeoff between payload and the number of requests. Please don’t write loops to crawl resources from id 0 to id 1000. Don’t pull resources too often, they usually do not change within a second and you will get banned upon detection.